A Glance at the Evolution of Cooperation from Parrondo's Point of View

Bachelor-Thesis
- Final Draft -

vorgelegt am: 9. September 2015

Wirtschaftsuniversität Wien

Name:	Friedrich Decker
Matrikelnummer:	0625081
Betreuer:	ao.Univ.Prof. Dr. Walter Böhm

Inhaltsverzeichnis

1 Einleitung

1.1 Problemstellung

Das Parrondo Paradoxon ist das prominenteste Beispiel einer Klasse paradox erscheinender Verhaltensweisen von Systemen, die sich unter bestimmten Voraussetzungen ergeben können, wenn die Kombination zweier Dynamiken innerhalb des Systems zu einem effizienteren Ergebnis führt als jede der beiden Einzeldynamiken[1]. Das als Parrondo Paradoxon bekannt gewordene Phänomen wurde vom spanischen Physiker Juan Parrondo in den 1990ern in Form eines Casinospiels modelliert, um das aus der Thermodynamik stammende Gedankenmodell eines brownschen Motors (bzw. brownsche Ratsche[2]) zu illustrieren[3].

Zwei stochastisch nicht unabhängige Spiele A und B werden Spieler G von einem Casino zum Spielen in beliebiger aber fester Reihenfolge angeboten. G besitzt ein unbegrenztes Vermögen, muss allerdings vor Spielantritt die - möglicherweise durch einen Zufallsmechanismus bestimmte - Reihenfolge, in der er A und B kombinieren will, sowie die Gesamtrundenanzahl bekannt geben. Spiel B besitzt in jeder Runde unveränderte Wahrscheinlichkeiten für Gewinn und Verlust, unabhängig vom Ausgang der Spiele in den Vorrunden (Bernoulli-Kette). Bei Spiel A hingegen sind die betreffenden Wahrscheinlichkeiten von der bisherigen Spielentwicklung (bzw. in der ersten Runde von einem Zufallsmechanismus) abhängig. Im ursprünglich von Parrondo vorgeschlagenen Spiel hängen diese Wahrscheinlichkeiten ausschließlich vom Kapital des Spielers modulo m (für ein kleines $m \in \mathbb{N}$) ab, sodass die stochastische Abhängigkeit zwischen den Spielen über das Vermögen hergestellt wird.

Beide Spiele seien isoliert betrachtet für G Verluststrategien, da G bei genügend großer Anzahl an Wiederholungen von (nur) Spiel A bzw. (nur) B einen für ihn ungünstigen Spielausgang und damit Verlust von Kapital zu erwarten hat. Unter bestimmten Voraussetzungen kann es für G dennoch - im Sinne eines positiven Erwartungswerts seines Profits - günstig sein, Spiel A und B in einer zuvor festgelegten Reihenfolge kombiniert zu spielen.

Die Annahme, dass das Vermögen des Spielers unbegrenzt sei, dient dazu, das Problem des Ruins des Spielers[4] im Modell zu eliminieren. Denn in der Realität besitzt das Casino

---

[1] Vgl. etwa [Roca et al., 2009, S. 2]

[2] Eine brownsche Ratsche, die ohne Energiezufuhr von außen funktioniert, wäre ein Perpetuum Mobile zweiter Art und verstieße somit gegen den 1. Hauptsatz der Thermodynamik, weshalb ein derartiges Gerät in der Realität nicht existieren kann.

[3] Vgl. [Harmer and Abbott, 1999, S. 207f]

[4] Gebräuchlicher ist die engl. Bezeichnung Gambler's Ruin Problem. Vgl. bspw. [Edwards, 1983]

in Relation zum Spieler in der Regel eine erdrückende finanzielle Übermacht, die einen Bankrott des Casinos nahezu unmöglich macht. Anders ist im Allgemeinen die Situation des Spielers: Er verfügt nur über ein relativ kleines Vermögen und ist zur Beendigung des Spiels gezwungen, sobald dieses verbraucht ist. Werfen Spieler X und Y mit den jeweiligen Vermögen V_X und V_Y eine faire Münze solange, bis einer der Spieler kein Vermögen mehr besitzt, und gewinnt X beim Ereignis „Kopf" und Y beim Ereignis „Zahl" jeweils eine Geldeinheit vom anderen Spieler, dann betragen die Wahrscheinlichkeiten für den Gewinn des gesamten Spiels $P(Xgewinnt) = \frac{V_X}{V_X+V_Y}$ und $P(Ygewinnt) = \frac{V_Y}{V_X+V_Y}$. Die gewonnenen Ergebnisse im Zuge der Untersuchung von Spielen in der vorliegenden Arbeit sollen allerdings nicht von der Relation der finanziellen Schlagkraft einzelner Spieler bzw. des Casinos abhängen, sondern allgemeiner Natur sein, weshalb unbegrenzte Vermögen für Spieler und Casino angenommen werden. Darüber hinaus soll mit der Bedingung, dass die Rundenanzahl zu Beginn des Spiels fest bestimmt wird, taktisches Verhalten des Spielers ausgeschlossen werden. Es soll ihm damit unmöglich gemacht werden, das Spiel nach Ausnutzung einer für ihn vorteilhaften Situation ad hoc zu verlassen.

Ein konkretes Beispiel[5], das im Folgenden präsentiert wird, dient zur Illustration der Ausgangslage und als Grundlage für die Bildung eines Modells[6] im nächsten Hauptteil der Arbeit: Ein Casino bietet ein, aus zwei Sub-Spielen A und B zusammengesetztes, Spiel auf Grundlage von „Europäischem Roulette"[7] an. Es sei angenommen, dass jeder der 37 möglichen Spielausgänge gleich wahrscheinlich ist. Spiel B ähnelt einem klassischen Roulettespiel: Der Spieler kann sich für eine der beiden Farben rot oder schwarz entscheiden und eine Geldeinheit auf diese wetten, um gegebenenfalls den doppelten Einsatz zu gewinnen. Da es sich um einen „fairen" Tisch handelt, und es jeweils 18 rote bzw. schwarze Fächer gibt, beträgt die Wahrscheinlichkeit zu gewinnen $\frac{18}{37}$, und damit ist die Wahrscheinlichkeit zu verlieren gleich $\frac{19}{37}$. Spiel B besitzt daher für den Spieler einen negativen Erwartungswert. Spiel A ist für ein Casino-Spiel weniger gewöhnlich: Ist das Spielkapital des Spielers ein Vielfaches von 3, gewinnt er, falls 1, 2 oder 3 gezogen wird, eine zusätzliche Geldeinheit und verliert andernfalls. Wenn das Spielkapital allerdings kein Vielfaches von 3 ist, gewinnt der Spieler bei jeder Zahl zwischen 1 und 28 (inklusive) und verliert andernfalls. Der Spieler kann vor Beginn des Spiels die Reihenfolge, in der er die Sub-Spiele spielen möchte, angeben. Es ist leicht einzusehen, dass dies notwendig ist, da er ansonsten immer nur dann Spiel A wählen würde, wenn dies der Situation entsprechend für ihn vorteilhaft ist, während er in den übrigen Runden durch

[5]Vgl. zum Aufbau des Spiels [Schilling, 2008].

[6]Dieses Modell wird im Folgenden als Casino-Modell bzeichnet.

[7]37 Fächer mit den Indizes 0 bis 36. Die Hälfte der Fächer von 1 bis 36 ist schwarz, die andere Hälfte rot. Das Fach mit der Nummer 0 ist grün.

die Wahl von Spiel B nur mit verhältnismäßig geringer Wahrscheinlichkeit einen Verlust in Kauf nehmen müsste. Wie im nächsten Hauptteil der Arbeit gezeigt wird, besitzt auch Spiel A einen negativen Erwartungswert. Dennoch sollte sich der Spieler überlegen, am Spiel teilzunehmen, da er durch Kombination der beiden Sub-Spiele unter bestimmten Voraussetzungen ein Spiel mit positivem Erwartungswert generieren kann.

Auch im zweiten Hauptteil der Arbeit werden Spiele betrachet. Allerdings handelt es sich bei diesen Spielen dann nicht, wie im ersten Hauptteil, um Zufallsexperimente, sondern um strategische Konflikte[8], die paarweise zwischen Spielern in einem Netzwerk bestehen. Die Spieler innerhalb des Netzwerks sind in der Lage, ihre erfolgreicheren Nachbarn, mehr oder weniger gekonnt, zu imitieren. Unterliegt die Qualität der Imitation einem stochastischen Prozess, kann dies unter gewissen Voraussetzungen dazu führen, dass die Effektivität des Gesamtsystems[9] stark gesteigert wird.

In beiden Teilbereichen der Arbeit wird es darum gehen, zwei ungünstige Dynamiken so miteinander zu kombinieren, dass eine günstige Dynamik entsteht. Schlussendlich werden Gemeinsamkeiten und Unterschiede zwischen dem Casino-Modell und dem Netzwerk-Simulationsmodell behandelt.

1.2 Faires Spiel und Glücksspielsysteme

Bei der Betrachtung von Spielen[10] stellt sich die Frage, welche Spiele für einen Spieler günstig bzw. ungünstig sind. Die Risikoeinstellung des Spielers bleibt dabei unberücksichtigt. Daher impliziert die Aussage, dass ein Spiel für einen Spieler günstig bzw. ungünstig ist, keine Aussage darüber, ob der Spieler eine Teilnahme am Spiel präferiert. Im Folgenden werden die Kennzeichen eines günstigen bzw. ungünstigen sowie eines fairen Spiels untersucht.

Betrachtet werde ein Spiel, dem ein Bernoulli-Prozess zugrunde liegt, mit der Zufallsvariable X und den Wahrscheinlichkeiten $p = P(X = 1)$ und $1 - p = P(X = 0)$[11]. X_k sei der Wert der Zufallsvariable X in Spielrunde k, die die jeweiligen Einnahmen des Spielers je Runde angibt. Die Partialsumme $S_n = \sum_{k=1}^{n} X_k$ bezeichnet die kumulativen Einnahmen des Spielers nach n Runden. Für die Teilnahme an jeder einzelnen Spielrunde hat der Spieler einen Eintrittspreis in Höhe von μ' zu entrichten. Der Nettogewinn

[8] Spiele werden in diesem Kontext als interdependente Mehrpersonen-Entscheidungsprobleme im Sinne der Spieltheorie aufgefasst.

[9] Die Gesamtwohlfahrt innerhalb des Netzwerks wird durch ein hohes Ausmaß an Kooperation zwischen den Spielern gesteigert.

[10] Spiele werden zunächst als Zufallsexperimente aufgefasst, auf deren Ausgang eine Wette abgeschlossen werden kann.

[11] Vgl. hierzu die Ausführungen über faire Spiele [Feller, 1968, S. 249ff]

des Spielers nach n Runden ist damit $S_n - n\mu'$. Sei $\mu = E(X_k)$ der Erwartungswert der Zufallsvariable X[12]. Falls $\mu \in \mathbb{R}$ gilt, ist das Gesetz der großen Zahlen anwendbar. Dieses ist formal definiert durch

$\forall \epsilon > 0 \ \exists n_0 \in \mathbb{N} \ \forall n > n_0 : P(| \ \frac{1}{n} \sum_{k=1}^{n} (X_k - E(X_k)) \ |> \epsilon) \to 0.$

Das Gesetz der großen Zahlen besagt nun, dass die Wahrscheinlichkeit für eine beliebig kleine Abweichung des durchschnittlichen Erfolgs des Bernoulli-Experiments vom Erwartungswert μ für ein hinreichend großes n gegen 0 konvergiert[13]. Daher ist der Betrag der Differenz $S_n - n\mu$ für ein hinreichend großes n mit großer Wahrscheinlichkeit klein im Verhältnis zu n[14], und falls $\mu' < \mu$ gilt, macht der Spieler bei einer genügend großen Anzahl an Spielrunden wahrscheinlich einen Gewinn in der Größenordnung $n(\mu - \mu')$. Ein solches Spiel wird als günstig für den Spieler bezeichnet. Ein Spiel, bei dem $\mu' > \mu$ gilt, führt durch analoge Argumentation zu einem Verlust in der Größenordnung von $n(\mu - \mu')$ und ist daher für den Spieler ungünstig[15].

Ein Spiel, für das $\mu' = \mu$ gilt, wird oftmals als faires Spiel bezeichnet. Diese Bezeichnung suggeriert, dass ein solches Spiel weder günstig noch ungünstig für den Spieler sei. Dass dies jedoch nicht der Fall sein muss, und ein sogenanntes faires Spiel auch ungünstig für einen Spieler sein kann, soll an folgendem Beispiel[16] gezeigt werden.

In jeder Spielrunde kann mit Wahrscheinlichkeit $(p_k)_{k \geq 0}$ mit $p_0 = 1 - (p_1 + p_2 + \ldots), p_k = \frac{1}{2^k k(k+1)}$ für $k \geq 1$ der Betrag $(x_k)_{k \geq 0}$ mit $x_0 = 0, x_k = 2^k$ für $k \geq 1$ gewonnen werden. Dann beträgt der erwartete Gewinn je Runde $\mu = \sum_{k \geq 0} x_k p_k = (1 - \frac{1}{2}) + (\frac{1}{2} - \frac{1}{3}) + (\frac{1}{3} - \frac{1}{4}) + \cdots = 1$ und die Varianz $\sigma^2 = \sum_{k \geq 0} (x_k - \mu)^2 p_k = \log(2) + \frac{1}{4} + \frac{3}{8} + \frac{49}{96} + \cdots = \infty$[17]. Unter der Annahme, dass der Spieler für jede Runde einen Eintrittspreis von 1 zahlt, liegt ein faires Spiel im Sinne von $\mu' = \mu$ vor, und der Nettogewinn nach n Runden beträgt $S_n - n$. Obwohl dieses Spiel als fair bezeichnet wird, kann gezeigt werden, dass $\forall \epsilon > 0 : P(S_n - n < -\frac{(1-\epsilon)n}{log_2 n}) \to 1$ gilt, und das Spiel somit ungünstig für den Spieler ist, da er für ein hinreichend großes n fast sicher einen entsprechenden Verlust erleiden wird.

Der Zentrale Grenzwertsatz besagt, dass die Summe der Realisierungen einer unabhängig verteilten Zufallsvariable mit wohldefiniertem Erwartungswert und wohldefinierter Standardabweichung für hinreichend große Stichproben unabhängig von der Verteilung,

[12] Im Fall $X \in \{0, 1\}$ gilt $\mu = p$.

[13] Vgl. etwa [Feller, 1968, S. 152]

[14] $P(| \ \frac{1}{n} \sum_{k=1}^{n} (X_k - E(X_k)) \ |< \epsilon) = P(| \sum_{k=1}^{n} (X_k - \mu) \ |< n\epsilon) = P(| \sum_{k=1}^{n} (X_k) - \sum_{k=1}^{n} (\mu) \ |< n\epsilon) = P(| \ S_n - n\mu \ |< n\epsilon) = P(| \ \frac{S_n}{n} - p \ |< \epsilon)$ und $\lim_{n \to \infty} P(| \ \frac{S_n}{n} - p \ |< \epsilon) \to 1.$

[15] Vgl. [Feller, 1968, S. 249]

[16] Entnommen aus [Feller, 1968, S. 262, 'Example of an unfavorable "fair" game.'].

[17] $\frac{(2^k - 1)^2}{2^k k(k+1)} = \frac{(2^k)^2 - 2^{k+1} + 1}{2^k k(k+1)} = \frac{(2^k)^2}{2^k k(k+1)} - \frac{2^{k+1}}{2^k k(k+1)} + \frac{1}{2^k k(k+1)} = \frac{2^k}{k(k+1)} - \frac{2}{k(k+1)} + \frac{1}{2^k k(k+1)} \to \infty$

die der Zufallsvariable zugrunde liegt, normalverteilt ist. Formal bedeutet das[18]:

Sei S_n die n-te Partialsumme einer Reihe bestehend aus den Realisierungen einer unabhängig und identisch verteilten Zufallsvariable mit dem Erwartungswert μ und der Standardabweichung $\sigma > 0$, $\mu, \sigma \in \mathbb{R}$. Dann gilt $Z_n = \frac{S_n - n\mu}{\sigma\sqrt{n}}$ und $\lim_{n\to\infty} Z_n \sim \mathcal{N}(0,1)$.

Im oben angeführten Beispiel gilt jedoch $\sigma^2 = \infty \iff \sigma = \infty \implies \sigma \notin \mathbb{R}$. Folglich kann in diesem Fall nicht davon ausgegangen werden, dass der Nettogewinn für große n annähernd gleichmäßig um 0 verteilt ist. Ein derartiges Spiel als fair zu bezeichnen, kann daher irreführend sein[19]. Falls $\sigma^2 \in \mathbb{R}$ gilt, scheint die Bezeichnung fair im Allgemeinen angebracht zu sein. Doch ist auch in diesen Fällen Vorsicht mit der Interpretation des Fairness-Begriffs geboten: Damit das Gesetz der großen Zahlen seine Bedeutung erlangt, muss eine sehr große Anzahl an Spielrunden vorliegen - immerhin handelt es sich um eine Grenzbetrachtung. Ein großes Versicherungsunternehmen, das jährlich Millionen von Versicherungen eines bestimmten Typs verkauft (beispielsweise KFZ-Versicherungen), ist mit dem Gesetz der Großen Zahlen konfrontiert. Bietet es eine faire Versicherung an, so wird der Gewinn im Verhältnis zum Gesamtumsatz mit großer Wahrscheinlichkeit nahe bei 0 liegen und ist mit annähernd gleicher Wahrscheinlichkeit positiv oder negativ. Für den Versicherungsnehmer, der einmal jährlich seinen Vertrag verlängert, ist das Gesetz der Großen Zahlen jedoch bedeutungslos. Mit großer Wahrscheinlichkeit wird er einen Verlust in Höhe der Versicherungsprämie erleiden, da kein Versicherungsfall auftritt[20].

Im nächsten Teil der Arbeit werden Spiele analysiert, die für den Spieler ungünstig sind. Diese Spiele sind nicht fair - ebenso wenig, wie günstige Spiele fair sind. Jedoch darf daraus nicht gefolgert werden, dass faire Spiele weder günstig noch ungünstig sein können. Simulationen, die im Laufe dieser Arbeit durchgeführt werden, basieren auf einer großen Anzahl von Wiederholungen und Stichproben, sodass das Gesetz der großen Zahlen in diesen Fällen von Bedeutung ist.

Seit Anbeginn des Glücksspiels versuchen Spieler, Strategien zu entwickeln, um andere Spieler bzw. das Casino zu besiegen. Wird der Ausgang eines Spiels zum Zeitpunkt i durch ein unabhängiges Ereignis A_i beschrieben, dann gilt $P(A_i) = P(A_i \mid A_{i-1} \cap A_{i-2} \cap \dots)$. Dies trifft etwa auf das Roulettespiel im Casino zu: Der Ausgang der Vorrunden beeinflusst die Wahrscheinlichkeiten in der aktuellen Runde nicht[21]. Dieser Umstand wird von manchen Spielern nicht erkannt. Sie glauben daran, dass Ergebnisse

[18] Vgl. hierzu etwa [Durrett, 2010, S. 106]

[19] Vgl. zu Interpretation des Fairness-Begriffs [Feller, 1968, S. 250f]

[20] Vgl. [Feller, 1968, S. 251]

[21] Abgesehen von der Möglichkeit, die Mechanik und die Charakteristik eines spezifischen Roulettetisches und das Verhalten eines Croupiers zu „lesen“, sodass äußerst erfahrene Spieler ihre A-priori-Wahrscheinlichkeiten revidieren können. Dieser Fall wird im Folgenden nicht beachtet.

der letzten Runden einen Einfluss auf die Wahrscheinlichkeiten der nachfolgenden Runden haben und versuchen, auf dieser Grundlage Spielstrategien zu entwickeln. Dies wird als Spielerfehlschluss bezeichnet[22].

Ein Glücksspielsystem stellt eine Spielstrategie dar, die aus einer Menge von festen Regeln aufgebaut ist, die jene Spielrunden bestimmen, in denen der Spieler eine Wette platziert. Beispielsweise kann eine derartige Regel beim Roulette lauten: Setze jede 5. Runde auf schwarz, oder setze dann auf rot, wenn in den letzten 7 Runden schwarz realisiert wurde.

Ein Random Walk ist ein stochastischer Prozess in diskreter Zeit mit unabhängigen und identisch verteilten Zuwächsen[23]. Er wird wie folgt definiert:

Sei $(Z_n)_{n\in\mathbb{N}}$ eine Folge von Realisierungen einer unabhängig und identisch verteilten Zufallsvariable $Z \in \mathbb{R}^d$ und $X_0 \in \mathbb{R}^d$ eine Konstante. Der durch $X_n = X_0 + \sum_{j=1}^{n} Z_j$ beschriebene stochastische Prozess $(X_n)_{n\in\mathbb{N}}$ wird d-dimensionaler Random Walk genannt.

Das sequentielle Spielen von Spiel B stellt einen deartigen eindimensionale Random Walk mit $X_0, Z_j \in \mathbb{N}$ dar. Bei Spiel A bzw. der Kombination von Spiel A und B ist hingegen die Unabhängigkeitsbedingung verletzt. Im Folgenden soll gezeigt werden, dass es unmöglich ist, auf Basis eines Random Walks ein Glücksspielsystem zu errichten.

Hierzu wird eine unendliche Folge von Bernoulli-Versuchen betrachtet, die als Spielrunden interpretiert werden können. In jeder Runde hat der Spieler die Wahl, auf den Ausgang des Experiments einen festen Betrag zu wetten[24] oder auszusetzen. Wenn es kein erfolgreiches Glücksspielsystem gibt, muss es egal sein, in welchen Runden der Spieler wettet: Der Spieler kann seine Ausgangssituation durch die Anwendung eines Glücksspielsystems nicht gegenüber der Situation, in der er in jeder Runde wettet, verbessern. Eine etwas formalere Definition für den Begriff des Glücksspielsystems lautet: [25]

Ein Glücksspielsystem ist eine Menge fester Regeln, die eindeutig für jede Runde bestimmen, ob ein Spieler in dieser Runde eine Wette platziert. In Runde k hängt diese Entscheidung nun ausschließlich von den Ergebnissen der $k-1$ Vorrunden ab und ist unabhängig von den Ausgängen der Runden $k, k+1, \ldots$. Die Regeln müssen so ausgestaltet sein, dass das Spiel unbegrenzt fortgesetzt wird[26].

Sei $P(s > r \mid n)$ die Wahrscheinlichkeit, dass der Spieler in n Runden öfter als r mal wettet. Da die Regeln des Systems fest sind, ist dieses Ereignis wohldefiniert, und

[22] Gebräuchlicher ist die engl. Bezeichnung Gambler's fallacy.

[23] Vgl. zu Random Walks [Feller, 1968, S. 73ff]

[24] Ist der Spieler in der Lage, den Wetteinsatz zu variieren, gibt es erfolgreiche und weniger erfolgreiche Strategien. Zu denken ist z.B. an die Martingale Strategie.

[25] Vgl. zur Definition von Glücksspielsystemen [Feller, 1968, S. 199]

[26] Die Regel, nach einer bestimmten Anzahl von Spielen, Gewinnen oder Verlusten das Spiel zu verlassen, wird nicht betrachtet.

die Wahrscheinlichkeit berechenbar. Zudem gilt $\lim_{n\to\infty} P(s > r \mid n) = 1$. Ohne Beschränkung der Allgemeinheit werde im Folgenden als Spiel exemplarisch ein Münzwurf betrachtet mit der Ereignismenge $\{„Kopf“, „Zahl“\}$ und $p = P(„Kopf“) = \frac{1}{2}$.

A_k bezeichne das Ereignis, dass die erste Wette in Runde k platziert wird. Da das Spiel unbegrenzt fortgesetzt wird, gilt: $\sum_{k\geq 1} P(A_k) = 1$. B_k sei das Ereignis „Kopf beim k-ten Versuch“ mit der Wahrscheinlichkeit $P(B_k) = p$ und B das Ereignis „Kopf bei der ersten Wette“. Dann ist $B = A_1B_1 \cup A_2B_2 \cup A_3B_3 \cup \ldots$. Nun hängt A_k nur von den Ausgängen der Runden $k-1, k-2, \ldots$ ab, und B_k ist unabhängig von allen Ausgängen in den Runden $\neq k$. Daher sind A_k und B_k voneinander unabhängig, und es gilt: $P(A_kB_k) = P(A_k)P(B_k) = \frac{1}{2}P(A_k)$. Daraus folgt: $P(B) = \sum_{k\geq 1} P(A_kB_k) = \frac{1}{2}\sum_{k\geq 1} P(A_k) = \frac{1}{2}$. Damit ist gezeigt, dass die Wahrscheinlichkeit für „Kopf bei der ersten Wette“ in jedem Glücksspielsystem $\frac{1}{2}$ beträgt[27]. Nun muss noch gezeigt werden, dass die Wetten stochastisch unabhängig sind. Sei A_k^* das Ereignis „zweite Wette beim k-ten Versuch“ und E das Ereignis „Kopf bei beiden Wetten“. Dann ist $E = \bigcup_{j<k} A_jB_jA_k^*B_k$ und $P(E) = \sum_{j=1}^{\infty}\sum_{k=j+1}^{\infty} P(A_jB_jA_k^*B_k)$. Weil B_k unabhängig von $A_jB_jA_k^*$ ist[28], gilt: $P(E) = \frac{1}{2}\sum_{j=1}^{\infty}\sum_{k=j+1}^{\infty} P(A_jB_jA_k^*) = \frac{1}{2}\sum_{j=1}^{\infty} P(A_jB_j)\sum_{k=j+1}^{\infty} P(A_k^* \mid A_jB_j)$. Wegen $P(A_k) \to 1$ und der Unbegrenztheit des Spiels gilt für ein festes A_jB_j mit $P(A_jB_j) > 0$ auch $\sum_{k=j+1}^{\infty} P(A_k^* \mid A_jB_j) = 1$. Daher gilt $P(E) = \frac{1}{4}$, und die Argumentation kann auf ein beliebiges Ereignis ausgedehnt werden[29].

□

[27] Diese Aussage überträgt sich induktiv auf „Kopf bei der k-ten Wette“.

[28] $A_jB_jA_k^*$ hängt nur von den Ausgängen der ersten k-1 Versuche ab.

[29] Der Beweis für die Unmöglichkeit eines erfolgreichen Glücksspielsystems wurde entnommen aus [Feller, 1968, S. 199f]

2 Casino-Modell

2.1 Diskrete Markow-Ketten

Ein diskreter stochastischer Prozess lässt sich anhand eines gerichteten Baums darstellen. Ein gerichteter Baum ist ein kreisfreier Graph, der aus Knoten und Kanten (Äste des Baums) besteht. Dabei symbolisieren die Knoten den Ausgang eines Einzelexperiments der Stufe n, und die gerichteten Kanten verbinden je zwei mögliche Ausgänge von Stufe n nach Stufe $n+1$ miteinander. Entlang eines Astes wird die Wahrscheinlichkeit notiert, mit der ein bestimmter Ausgang auf Stufe $n+1$ realisiert wird, wenn der Ausgang des Experiments auf Stufe n feststeht. Äste, die unmöglich sind (Wahrscheinlichkeit von null), werden weggelassen. Daraus ergibt sich, dass jeder Knoten höchstens einen direkten Vorgängerknoten hat. Ein realisierbarer Weg durch einen Baum vom Ursprung bis zu einem Endknoten steht für den Ausgang des Gesamtexperiments, das sich aus $m \in (\mathbb{N} \cup \infty)$ Einzelexperimenten zusammensetzt, und wird als Pfad bezeichnet. Durch die Auflistung $\mathbf{U}$ aller möglichen Pfade t_i und der dazugehörigen Pfadwahrscheinlichkeiten $w(t_i)$ ist der Baum vollständig gegeben. Dabei stellt t_i ein m-Tupel dar, das eine der möglichen Sequenzen von Einzelexperimentausgängen beinhaltet. Es ist möglich, eine Folge von Funktionen $(f_n)_{n=0,\ldots,m}$ zu definieren. Die Grundmenge von f_n ist der Baum der Stufe n, T_n, und die Zielmenge U_n besteht aus allen möglichen Ausgängen eines Einzelexperiments der Stufe n[30].

Es sei $S = \{s_1, s_2, s_3, ...\}$ ein endlicher oder abzählbar unendlicher Zustandsraum, der die möglichen Ausgänge aller Einzelexperimente (Zustände) beinhaltet. Ein diskreter stochastischer Prozess heißt unabhängiger Prozess, wenn $P(f_{n+1} = s_{j_{n+1}} \mid f_n = s_{j_n}, f_{n-1} = s_{j_{n-1}}, \ldots, f_0 = s_{j_0}) = P(f_{n+1} = s_{j_{n+1}})$ gilt, und Markow-Prozess, falls $P(f_{n+1} = s_{j_{n+1}} \mid f_n = s_{j_n}, f_{n-1} = s_{j_{n-1}}, \ldots, f_0 = s_{j_0}) = P(f_{n+1} = s_{j_{n+1}} \mid f_n = s_{j_n})$[31]. Bei einem unabhängigen Prozess beeinflussen die Realisierungen in den vorangegangenen Zufallsexperimenten somit die Einschätzung über den Ausgang des aktuell durchgeführten Experiments nicht. Bei einem Markow-Prozess wird diese Bedingung etwas gelockert, sodass nur der (exakte[32]) Ausgang des unmittelbar zuvor durchgeführten Zufallsexperiments die Wahrscheinlichkeiten des aktuellen Experiments beeinflusst[33]. Der Umstand, dass mit dem Wissen über den Ausgang des unmittelbar vorausgehenden Experiments

[30] Vgl. [Kemeny and Snell, 1976, S. 15f]

[31] Vgl. [Kemeny and Snell, 1976, S. 24]

[32] Der Ausgang muss einem einzelnen Element des Zustandsraums zuzuordnen sein. Bezieht sich der Ausgang bzw. das Wissen über den Ausgang auf eine Teilmenge des Zustandsraums, sind die Ausgänge in den Runden davor relevant.

[33] Generell ist es auch möglich, den Ausgang einer fixen Anzahl zuvor durchgeführter Zufallsexperimente als Bedingung zu betrachten.

jegliche Information über Ausgänge in den Runden davor für die Bewertung der Wahrscheinlichkeiten in der aktuellen Runde nutzlos ist, wird als Markow-Eigenschaft bezeichnet.

Die Übergangswahrscheinlichkeiten für einen Markow-Prozess der Stufe n sind $p_{ij}(n) = P(f_n = s_j \mid f_{n-1} = s_i)$. Eine endliche Markow-Kette ist ein endlicher Markow-Prozess, bei dem die Übergangswahrscheinlichkeiten nicht von der Anzahl der Wiederholungen abhängen, also $p_{ij}(n) = p_{ij}$ gilt[34]. Die Übergangsmatrix für eine Markow-Kette wird mit $\mathbf{P}$ bezeichnet und enthält alle Einträge p_{ij}. Ein stochastischer Startvektor π_0 ist ein Zeilenvektor, der die Startverteilung angibt und alle $p_j^{(0)} = P(f_0 = s_j)$ beinhaltet[35]. Eine Markow-Kette wird vollständig durch einen stochastischen Startvektor π_0 und eine Übergangsmatrix $\mathbf{P}$ determiniert.

Wegen $\pi_n = \pi_{n-1}\mathbf{P} = \pi_0\mathbf{P}^n$ stehen bei der Untersuchung von Markow-Ketten zwei Fragen im Mittelpunkt: Welche Auswirkungen hat eine Änderung von π_0 auf π_n? Wie verhält sich $\mathbf{P}^n$ für $n = 2, 3, \ldots$ und $n \to \infty$? π_n gibt die PMF[36] über dem Zustandsraum gemäß der Übergangsmatrix $\mathbf{P}$ für einen n-stufigen Prozess an. Die Matrix $\mathbf{P}^n$ wird folgendermaßen interpretiert: Die ij-te Komponente von $\mathbf{P}^n$ gibt die Wahrscheinlichkeit an, sich nach n Schritten in Zustand j zu befinden, falls zum Zeitpunkt t = 0 in Zustand i gestartet wurde[37].

Diverse Eigenschaften der Übergangsmatrix bzw. des stochastischen Startvektors sind von Interesse, da sie einerseits Aufschluss über das Verhalten stochastischer Prozesse geben und andererseits bestimmen, auf welche Weise sich eine Markow-Kette untersuchen lässt. Im Folgenden werden Definitionen gegeben, die im weiteren Verlauf dazu dienen werden, diese Eigenschaften zu beschreiben und schlussendlich eine Klassifizierung von Markow-Ketten vorzunehmen.

Sei T eine schwache Ordnungsrelation und ($\mathbf{P}$,π_0) eine Markow-Kette auf dem Zustandsraum $S = \{s_1, s_2, s_3, \ldots\}$, wobei $s_i T s_j$ bedeutet, dass, gemäß $\mathbf{P}$, Zustand s_j von Zustand s_i aus mit positiver Wahrscheinlichkeit erreichbar ist, bzw. $s_i = s_j$ gilt. Anders ausgedrückt existiert ein Pfad von s_i nach s_j, der nur Kanten mit positiver Wahrscheinlichkeit aufweist. Aus T ergibt sich eine Äquivalenzrelation R mit $s_i R s_j \iff (s_i T s_j \wedge s_j T s_i)$. R partitioniert S in Äquivalenzklassen, wobei innerhalb einer Äquivalenzklasse jeder Zustand von jedem anderen Zustand aus erreicht werden kann[38]. Insbesondere kann jeder Zustand innerhalb einer Äquivalenzklasse der Mächtigkeit größer 1

[34]Vgl. [Kemeny and Snell, 1976, S. 25]
[35]Vgl. [Kemeny and Snell, 1976, S. 33]
[36]Probability Mass Function bzw. Wahrscheinlichkeitsfunktion.
[37]Vgl. [Kemeny and Snell, 1976, S. 34]
[38]Vgl. [Kemeny and Snell, 1976, S. 35]

sich selbst über zumindest einen anderen Zustand der selben Äquivalenzklasse erreichen. Daneben induziert T eine partielle Ordnung bzw. Halbordnung T^* folgendermaßen: Seien $u, v \subseteq S$ zwei Äquivalenzklassen. Dann bedeutet uT^*v, dass von u aus alle Elemente von v erreichbar sind, jedoch von v aus keines der Elemente in u erreichbar ist, außer für den Fall, dass u=v gilt. T^* zeigt an, zu welchen Zuständen sich ein stochastischer Prozess mit (relativ) großer Wahrscheinlichkeit bewegen wird[39].

Sei A eine nichtleere Menge und T' eine Quasiordnung[40] auf A. Dann gilt: $\exists a \in A \forall x \in A : aT'x \Rightarrow xT'a$, wobei a minimales Element genannt wird. Darüber hinaus gilt: $\exists b \in A \forall x \in A : xT'b \Rightarrow bT'x$, und b ist das maximale Element. Anders ausgedrückt: Eine nichtleere Menge, auf der eine Quasiordnung definiert ist, besitzt bezüglich dieser Quasiordnung zumindest ein minimales und zumindest ein maximales Element, die nicht notwendigerweise unterschiedlich sein müssen[41].

Weil eine Halbordnung immer auch eine Quasiordnung ist, gilt: Eine Äquivalenzklasse bezüglich der - aus einer schwachen Ordnungsrelation T abgeleiteten - Äquivalenzrelation R, die das soeben genannte Minimalitätskriterium erfüllt, wird ergodisch genannt und existiert in jeder Markow-Kette. Wenn ein Zustand einer ergodischen Äquivalenzklasse erreicht wird, kann sie nie wieder verlassen werden. Wenn eine nicht-ergodische Äquivalenzklasse verlassen wird, kann sie nie wieder erreicht werden. Besitzt eine bezüglich **P** ergodische Teilmenge des Zustandsraums nur ein einziges Element s_i, so wird dieses Element absorbierender Zustand genannt, und es gilt: $p_{ii} = 1$[42].

Darüber hinaus wird eine weitere Unterteilung in zyklische und reguläre (also nicht-zyklische) Äquivalenzklassen vorgenommen (Aperiodizität). Anschaulich beschrieben kann, nach dem Verstreichen eines beliebigen Zeitraums hinreichender Länge, - von jedem beliebigen Startzustand ausgehend - jeder Zustand einer regulären Äquivalenzklasse mit positiver Wahrscheinlichkeit erreicht werden, während dies innerhalb einer zyklischen Äquivalenzklasse nicht möglich ist[43]. Betrachtet wird eine Äquivalenzklasse der Mächtigkeit m > 1. Nun existiert zumindest ein Pfad von jedem Zustand zu sich selbst, der über zumindest einen anderen Zustand (der selben Äquivalenzklasse) führt. Sei V die nichtleere Menge aller Pfade von s_i nach s_i, die nur über Zustände der betrachteten Äquivalenzklasse verlaufen und s_i beliebig aber fest innerhalb der betrachteten Äquivalenzklasse. Eine Äquivalenzklasse ist genau dann regulär, wenn $ggT(\{v \in V : |v|\}) = 1$. Andernfalls ist sie zyklisch.

[39]Vgl. [Kemeny and Snell, 1976, S. 5]
[40]Eine Quasiordnung ist eine reflexive und transitive Relation.
[41]Vgl. [Kemeny and Snell, 1976, S. 5]
[42]Vgl. [Kemeny and Snell, 1976, S. 35]
[43]Vgl. [Kemeny and Snell, 1976, S. 36]

Der Zustandsraum S werde nun gemäß der oben definierten Äquivalenzrelation R in Äquivalenzklassen $u_1, \ldots, u_k$ mit $k \geq 1$ partitioniert. Dabei gelte: u_1 sei ergodisch und $\forall l, m \in \mathbb{N} \setminus \{0, 1\} : l < m \Rightarrow (v \in u_l \wedge w \in u_m \Rightarrow p_{vw} = 0)$. Außerdem ist u_m ergodisch, falls $p_{wv} = 0$. Die Übergangsmatrix **P** ist äquivalent zur permutierten Übergangsmatrix **P'**, da der Zustandsraum eine Menge ist. Deshalb kann in jedem Fall folgende Übergangsmatrix gebildet werden:

$$\begin{array}{c} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{array} \begin{bmatrix} P_1 & & & & \\ R_2 & P_2 & & \mathbf{0} & \\ & R_3 & P_3 & & \\ & R_4 & & P_4 & \\ & & R_5 & & P_5 \end{bmatrix}$$

(a) Quelle: Kemeny et al., 1976, S. 36

Abbildung 1: Permutierte Übergangsmatrix, Einteilung in Äquivalenzklassen

Dabei bezeichnet P_i die Übergangsmatrix innerhalb einer Äquivalenzklasse. Die Wahrscheinlichkeit für einen Übergang von einem Zustand einer gegebenen Äquivalenzklasse zu einem Zustand einer Äquivalenzklasse höherer Stufe ist null. Ist u_i ergodisch, besteht die Matrix R_i, die die Übergangswahrscheinlichkeiten zu einer Äquivalenzklasse niedrigerer Stufe beinhaltet, ebenfalls ausschließlich aus Nullen. Ist u_i hingegen nicht-ergodisch, enthält R_i zumindest einen positiven Eintrag.

P' vereinfacht die Untersuchung einer Markow-Kette, da der mit **0** bezeichnete Bereich in der n-ten Potenz von **P'** wiederum ausschließlich aus Null-Einträgen besteht. Darüber hinaus können Äquivalenzklassen - in Form der Matrizen P_i - getrennt voneinander untersucht werden. Für ein hinreichend großes n gibt $(P_i)^n$ Aufschluss darüber, ob eine Äquivalenzklasse zyklisch ist, da in diesem Fall Komponenten mit Null-Einträgen vorhanden sein müssen, und, falls sie nicht-zyklisch ist, keine Null-Einträge vorhanden sein dürfen. Die n-ten Potenzen von R_i wiederum zeigen die Entwicklung der Wahrscheinlichkeit an, eine nicht-ergodische Äquivalenzklasse zu verlassen.

Folgende Fälle lassen sich nun unterscheiden[44]:

1. Markow-Ketten, die ausschließlich aus ergodischen Äquivalenzklassen bestehen:

[44] Vgl. zur Einteilung von Markow-Ketten [Kemeny and Snell, 1976, S. 36ff].

a) Es existiert genau eine ergodische Äquivalenzklasse: Die Markow-Kette wird ergodische Kette genannt.

- Die ergodische Äquivalenzklasse ist regulär: Hierbei handelt es sich um eine reguläre Markow-Kette, und alle Komponenten der n-ten Potenzen von $\mathbf{P}$ sind für ein hinreichend großes n positiv. Die Übergangsmatrix wird regulär genannt, und $\exists N_0 \in \mathbb{N} \forall n \geq N_0 : \mathbf{P}^n > \mathbf{0}$.
- Die ergodische Äquivalenzklasse ist zyklisch: Dann liegt eine zyklische Markow-Kette vor. Eine solche Markow-Kette besitzt eine Periode $d > 1$. Ihr Zustandsraum kann in d zyklische Teilmengen unterteilt werden.

b) Es existieren mehrere ergodische Äquivalenzklassen: Die Markow-Kette besteht aus mehreren, stochastisch voneinander unabhängigen Markow-Ketten. Jede (ergodische) Äquivalenzklasse stellt eine eigene Markow-Kette dar. Diese sollten dann getrennt und separat untersucht werden.

2. Markow-Ketten, die aus ergodischen und nicht-ergodischen Äquivalenzklassen bestehen:

a) Alle ergodischen Äquivalenzklassen sind einelementig: Die Markow-Kette wird in diesem Fall absorbierend genannt.

b) Alle ergodischen Äquivalenzklassen sind regulär, aber nicht alle sind einelementig.

c) Alle ergodischen Äquivalenzklassen sind zyklisch.

d) Es existieren reguläre und zyklische ergodische Äquivalenzklassen.

Die Übergangsmatrix $\mathbf{P}$ einer Markow-Kette kann keine, eine oder mehrere stationäre Wahrscheinlichkeitsverteilungen besitzen. Bezeichne $\bar{\pi} : \Sigma_i \bar{\pi}_i = 1 \Rightarrow \bar{\pi} \neq \vec{0}$ eine mögliche stationäre Wahrscheinlichkeitsverteilung einer endlichen Markow-Kette, die in Form eines Reihenvektors gegeben ist. Dann gilt: $\forall n \in \mathbb{N} : \bar{\pi}\mathbf{P}^n = \bar{\pi}$ bzw. $(\mathbf{P}^n)^T \bar{\pi}^T = \bar{\pi}^T$. Eine stationäre Wahrscheinlichkeitsverteilung ist also ein Links-Eigenvektor aller n-ten Potenzen der Übergangsmatrix zum Eigenwert $\lambda = 1$. Wie gezeigt werden kann, besitzt eine ergodische Markow-Kette mit endlichem Zustandsraum immer genau eine stationäre Wahrscheinlichkeitsverteilung[45]. Diese gibt für $n \to \infty$ die Grenzwerte der relativen Häufigkeiten an, mit der jeder Zustand des Zustandsraums realisiert wird.

Sei $\mathbf{P}$ eine reguläre Übergangsmatrix. Dann gilt $\lim_{n\to\infty} \mathbf{P}^n \to \mathbf{A}$, $\mathbf{A} = \xi\alpha$[46] und $\alpha >$

[45] Vgl. [Feller, 1968, S. 394ff]

[46] ξ ist ein Reihenvektor der Länge $|S|$, wobei alle Komponenten von ξ gleich 1 sind. α ist ein Zeilenvektor der Länge $|S|$.

$\vec{0}$. Jede Zeile in $\mathbf{A}$ entspricht dem Vektor α, und dieser ist identisch mit der stationären Wahrscheinlichkeitsverteilung $\bar{\pi}$. Für eine beliebige Anfangsverteilung π gilt $\lim\limits_{n\to\infty} \pi\mathbf{P}^n \to \bar{\pi}$. Die Anfangsverteilung spielt im Limit keine Rolle für die relativen Häufigkeiten, mit der die einzelnen Zustände realisiert werden.

2.2 Funktionsweise der Parrondo-Strategie

Nun ist es möglich, das unter 1.1 beschriebene Spiel A näher zu untersuchen. Bezeichne m das Kapital des Spielers und sei $S = \{s_0, s_1, s_2\}$, wobei s_i dem Zustand „$m \equiv i \mod 3$“ entspricht.

$$\mathbf{P} = \begin{array}{c|ccc} & s_0 & s_1 & s_2 \\ \hline s_0 & 0 & \frac{3}{37} & \frac{34}{37} \\ s_1 & \frac{9}{37} & 0 & \frac{28}{37} \\ s_2 & \frac{28}{37} & \frac{9}{37} & 0 \end{array}, \quad \lim_{n\to\infty} \mathbf{P}^n = \begin{array}{c|ccc} & s_0 & s_1 & s_2 \\ \hline s_0 & 0.389 & 0.145 & 0.466 \\ s_1 & 0.389 & 0.145 & 0.466 \\ s_2 & 0.389 & 0.145 & 0.466 \end{array}$$

$\mathbf{P}$ ist eine reguläre Übergangsmatrix. Daraus kann die stationäre Wahrscheinlichkeitsverteilung berechnet werden[47]. $\alpha = (0.389, 0.145, 0.466)$ gibt die relativen Häufigkeiten an, mit der die Zustände des Zustandsraums im Limit $n \to \infty$ auftreten. In Zustand s_0 beträgt die Wahrscheinlichkeit zu gewinnen $\frac{3}{37}$ und in Zustand s_1 und s_2 jeweils $\frac{28}{37}$, sodass sich eine Gewinnwahrscheinlichkeit von $p = 0.389 * \frac{3}{37} + (0.145+0.466) * \frac{28}{37} = 0.494 < \frac{1}{2}$ ergibt. Eine Simulation mit dem unter 'Simulation des Casino-Modells' angeführten Java-Programm bestätigt das Ergebnis. Spiel A ist also (wie Spiel B[48]) ungünstig für den Spieler.

Betrachtet werde nun das Spiel, das sich aus den Sub-Spielen A und B zusammensetzt, wobei A und B zufällig mit $P(A) = P(B) = \frac{1}{2}$ gespielt werden (probabilistische Strategie). $S = \{s_0A, s_1A, s_2A, s_0B, s_1B, s_2B\}$, und s_iX entspricht dem Zustand „$m \mod 3 \equiv i$, und es wird Spiel X gespielt“.

$$\mathbf{P} = \begin{array}{c|cccccc} & s_0A & s_1A & s_2A & s_0B & s_1B & s_2B \\ \hline s_0A & 0 & \frac{3}{74} & \frac{34}{74} & 0 & \frac{3}{74} & \frac{34}{74} \\ s_1A & \frac{9}{74} & 0 & \frac{28}{74} & \frac{9}{74} & 0 & \frac{28}{74} \\ s_2A & \frac{28}{74} & \frac{9}{74} & 0 & \frac{28}{74} & \frac{9}{74} & 0 \\ s_0B & 0 & \frac{18}{74} & \frac{19}{74} & 0 & \frac{18}{74} & \frac{19}{74} \\ s_1B & \frac{19}{74} & 0 & \frac{18}{74} & \frac{19}{74} & 0 & \frac{18}{74} \\ s_2B & \frac{18}{74} & \frac{19}{74} & 0 & \frac{18}{74} & \frac{19}{74} & 0 \end{array}$$

[47] Siehe Anhang, 'Berechnung der stationären Wahrscheinlichkeitsverteilung'.
[48] Die Gewinnwahrscheinlichkeit in Spiel B beträgt $\frac{18}{37} = 0.\overline{486}$.

Da $\mathbf{P}$ wieder eine reguläre Übergangsmatrix ist, wird wie oben vorgegangen, und es ergibt sich: $\alpha = (0.172858, 0.125392, 0.201750, 0.172858, 0.125392, 0.201750)$ und $p = 0.172858 * \frac{3}{37} + (0.125392 + 0.201750) * \frac{28}{37} + (0.172858 + 0.125392 + 0.201750) * \frac{18}{37} = 0.50482\overline{567} > \frac{1}{2}$. Das Ergebnis wird durch Simulation mit entsprechenden Parametern bestätigt. Die Gewinnwahrscheinlichkeit liegt über 0.5. Damit ist dieses Spiel für den Spieler günstig!

Als weiteres Beispiel werde das Spiel betrachtet, bei dem Sub-Spiel A und B abwechselnd gespielt werden. Ein Spiel, bei dem Spiel A und B gemäß einer periodischen Sequenz gespielt werden, wird im Folgenden als deterministische Strategie bezeichnet. Der Zustandsraum dieses Spiels gleicht dem soeben betrachteten Zustandsraum, und es ergibt sich die Übergangsmatrix

$$\mathbf{P} = \begin{array}{c} \\ s_0A \\ s_1A \\ s_2A \\ s_0B \\ s_1B \\ s_2B \end{array} \begin{array}{c} \begin{array}{cccccc} s_0A & s_1A & s_2A & s_0B & s_1B & s_2B \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & \frac{3}{37} & \frac{34}{37} \\ 0 & 0 & 0 & \frac{9}{37} & 0 & \frac{28}{37} \\ 0 & 0 & 0 & \frac{28}{37} & \frac{9}{37} & 0 \\ 0 & \frac{18}{37} & \frac{19}{37} & 0 & 0 & 0 \\ \frac{19}{37} & 0 & \frac{18}{37} & 0 & 0 & 0 \\ \frac{18}{37} & \frac{19}{37} & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Diese Übergangsmatrix ist ergodisch und zyklisch, da jeder Zustand von jedem Zustand aus erreicht werden kann, aber - wie bei jeder deterministischen Strategie - nicht zu jedem beliebigen Zeitpunkt erreichbar ist. Wenn beispielsweise in der ersten Runde mit Spiel A gestartet wird, dann wird Spiel A auch in der 3., 5., 7., ... Runde gespielt, nicht aber in der 2., 4., 6., ... Runde. Bei einer zyklischen Übergangsmatrix konvergiert $\mathbf{P}^n$ nicht; jedoch kann die stationäre Wahrscheinlichkeitsverteilung als Eigenvektor von $\mathbf{P}$ zum Eigenwert 1 gefunden werden.[49]

$\bar{\pi} = (0.188563, 0.236451, 0.074986, 0.114261, 0.033529, 0.352210)$ und $p = 0.188563 * \frac{3}{37} + (0.236451 + 0.074986) * \frac{28}{37} + (0.114261 + 0.033529 + 0.352210) * \frac{18}{37} = 0.494214\overline{189} < \frac{1}{2}$. Die Sequenz 'ABABAB...' ist also ungünstig für den Spieler, und dieser hat auf lange Sicht bei diesem Spiel mit einem Verlust zu rechnen.

Die Übergangsmatrizen für die Sequenzen 'AABAAB...' und 'BBABBA...' sind:

[49] Siehe Anhang, 'Berechnung der stationären Wahrscheinlichkeitsverteilung'.

$$\mathbf{P}_{AAB} =$$

$$\begin{array}{c|ccccccccc} & s_0AA & s_1AA & s_2AA & s_0AB & s_1AB & s_2AB & s_0BA & s_1BA & s_2BA \\ \hline s_0AA & 0 & 0 & 0 & 0 & \frac{3}{37} & \frac{34}{37} & 0 & 0 & 0 \\ s_1AA & 0 & 0 & 0 & \frac{9}{37} & 0 & \frac{28}{37} & 0 & 0 & 0 \\ s_2AA & 0 & 0 & 0 & \frac{28}{37} & \frac{9}{37} & 0 & 0 & 0 & 0 \\ s_0AB & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{18}{37} & \frac{19}{37} \\ s_1AB & 0 & 0 & 0 & 0 & 0 & 0 & \frac{19}{37} & 0 & \frac{18}{37} \\ s_2AB & 0 & 0 & 0 & 0 & 0 & 0 & \frac{18}{37} & \frac{19}{37} & 0 \\ s_0BA & 0 & \frac{3}{37} & \frac{34}{37} & 0 & 0 & 0 & 0 & 0 & 0 \\ s_1BA & \frac{9}{37} & 0 & \frac{28}{37} & 0 & 0 & 0 & 0 & 0 & 0 \\ s_2BA & \frac{28}{37} & \frac{9}{37} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

$$\mathbf{P}_{BBA} =$$

$$\begin{array}{c|ccccccccc} & s_0BB & s_1BB & s_2BB & s_0AB & s_1AB & s_2AB & s_0BA & s_1BA & s_2BA \\ \hline s_0BB & 0 & 0 & 0 & 0 & \frac{18}{37} & \frac{19}{37} & 0 & 0 & 0 \\ s_1BB & 0 & 0 & 0 & \frac{19}{37} & 0 & \frac{18}{37} & 0 & 0 & 0 \\ s_2BB & 0 & 0 & 0 & \frac{18}{37} & \frac{19}{37} & 0 & 0 & 0 & 0 \\ s_0AB & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{3}{37} & \frac{34}{37} \\ s_1AB & 0 & 0 & 0 & 0 & 0 & 0 & \frac{9}{37} & 0 & \frac{28}{37} \\ s_2AB & 0 & 0 & 0 & 0 & 0 & 0 & \frac{28}{37} & \frac{9}{37} & 0 \\ s_0BA & 0 & \frac{18}{37} & \frac{19}{37} & 0 & 0 & 0 & 0 & 0 & 0 \\ s_1BA & \frac{19}{37} & 0 & \frac{18}{37} & 0 & 0 & 0 & 0 & 0 & 0 \\ s_2BA & \frac{18}{37} & \frac{19}{37} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Beide Übergangsmatrizen sind ergodisch und zyklisch.
$\bar{\pi}_{AAB} = (.112451, .03262, .188261, .150403, .054911, .128020, .090477, .138909, .103947)$,
$p_{AAB} = (.112451 + .090477) * \frac{3}{37} + (.03262 + .188261 + .138909 + .103947) * \frac{28}{37} + (.150403 + .054911 + .128020) * \frac{18}{37} = .529552\overline{216}$.
$\bar{\pi}_{BBA} = (.105863, .147459, .080012, .114647, .092588, .126099, .117947, .039968, .175418)$,
$p_{BBA} = .114647 * \frac{3}{37} + (.092588 + .126099) * \frac{28}{37} + (.105863 + .147459 + .080012 + .117947 + .039968 + .175418) * \frac{18}{37} = .499113\overline{054}$.

Es wird nun noch das Spiel mit Sequenz 'BBBA' durch Simulation betrachtet. Auch dieses Spiel ist ungünstig für den Spieler, da die Simulation ein $p = 0.49827 < \frac{1}{2}$ ergibt. Es stellt sich nun die Frage, warum manche Strategien günstig sind, und andere ungünstig. Zur Klärung dieser Frage ist es notwendig, die Ursache zu verstehen, die dazu führt, dass zwei ungünstige Spiele in Kombination zu einem günstigen Ergebnis führen können.

Spiel A und Spiel B sind, wenn sie kombiniert gespielt werden, stochastisch nicht unabhängig, da in der Sequenz '…BA…' der Ausgang von Spiel B Einfluss auf die Gewinnwahrscheinlichkeit in Spiel A hat. Die Abhängigkeit zwischen Spiel A und B wird über das Vermögen des Spielers, das durch den Ausgang von Spiel B beeinflusst wird und wiederum Einfluss auf seine Gewinnwahrscheinlichkeit in Spiel A hat, hergestellt. Bei einer ersten naiven Herangehensweise an die Untersuchung von Spiel A könnte, unter der Annahme, dass alle Zustände gleich häufig auftreten, von einer Gewinnwahrscheinlichkeit von $\frac{1}{3} * \frac{3}{37} + \frac{2}{3} * \frac{28}{37} = 0.534$ ausgegangen werden. Wie oben bereits gezeigt wurde, tritt die Situation, dass das Vermögen des Spielers ein Vielfaches von 3 ist, allerdings öfter als $\frac{1}{3}$ der Zeit auf: Ist das Vermögen ein Vielfaches von 3, dann wird der Spieler mit sehr großer Wahrscheinlichkeit verlieren. Anschließend gewinnt er mit relativ großer Wahrscheinlichkeit. Daher ist etwa $\ldots s_0 s_2 s_0 s_2 \ldots$ als Teil einer Folge von Zuständen verhältnismäßig wahrscheinlich. Zustand s_1 wird hingegen von den beiden anderen Zuständen aus relativ selten erreicht: Von Zustand s_0 aus nur $\frac{3}{37}$ der Zeit und von Zustand s_2 aus nur $\frac{9}{37}$ der Zeit. Es muss nun ein Mechanismus gefunden werden, der den „schlechten“ Zustand s_0 vermeiden hilft und die „guten“ Zustände s_1 und s_2 wahrscheinlicher macht.

Betrachtet werde das Spiel, das sich ergibt, wenn die Sequenz 'AAB' periodisch gespielt wird - also im Vergleich zum soeben betrachteten Spiel in jeder dritten Runde Spiel A durch Spiel B ersetzt wird. Unabhängig davon, in welchem Zustand wir uns in den ersten beiden Perioden der Sequenz befinden, wissen wir, dass wir uns nach zwei Runden Spiel A in der dritten Periode mit relativ großer Wahrscheinlichkeit in einem der Zustände s_0AB oder s_2AB befinden. Falls wir uns in s_2AB befinden, stellt dies eine Verschlechterung gegenüber dem Zustand s_2AA dar, da dann nur mit $p = \frac{18}{37}$ anstatt mit $p = \frac{28}{37}$ gewonnen wird. Befinden wir uns jedoch in s_0AB, dann stellt dies gegenüber dem Zustand s_0AA einen sehr großen Vorteil dar. Denn nun beträgt die Wahrscheinlichkeit zu gewinnen $\frac{18}{37}$ anstatt wie zuvor nur $\frac{3}{37}$. Insgesamt überwiegt hier der positive Effekt, der sich daraus ergibt, bei einem durch 3 teilbaren Vermögen in jeder dritten Runde nicht Spiel A spielen zu müssen, sodass der Abwärtstrend umgekehrt wird. In Analogie zum thermodynamischen Ratschenexperiment kann Spiel B als - nicht perfekt arbeitende - Sperrvorrichtung betrachtet werden.

Da die Spiele A und B nicht stochastisch unabhängig sind, entsteht bei ihrer Kombination eine neue Wahrscheinlichkeitsverteilung. Der Erwartungswert dieser neuen Verteilung muss dann nicht zwischen den Erwartungswerten der beiden alten Verteilungen von Spiel A bzw. B liegen, sondern kann sich über oder unter diesen beiden Werten befinden. Im Zusammenhang mit dem hier betrachteten Spiel sind vor allem jene neuen Verteilungen interessant, deren Erwartungswert sich möglichst weit über den alten Erwartungswerten befindet.

Nun bleibt noch die Frage zu beantworten, warum die periodische Anwendung der Sequenz 'AB' ein $p < \frac{1}{2}$ liefert, die probabilistische Strategie mit $P(A) = P(B) = \frac{1}{2}$ jedoch ein $p > \frac{1}{2}$. Wird die stationäre Wahrscheinlichkeitsverteilung der deterministischen Strategie 'AB' betrachtet, so fällt auf, dass der „gute" Zustand s_2A sehr selten vorkommt, der Zustand s_2B allerdings sehr häufig. s_2A ist von s_1B und von s_0B aus direkt erreichbar. s_1B kommt allerdings nur äußerst selten vor, da die Wahrscheinlichkeit, von s_0A oder s_2A nach s_1B zu gelangen, sehr gering ist. Daher wird auch s_2A nur selten erreicht. s_2B wird hingegen von s_1A sowie s_0A sehr häufig erreicht. Weil in 'AB' der für den Spieler wünschenswerte Zustand s_2A relativ oft durch den weniger positiven Zustand s_2B ersetzt wird, ist das Spiel ungünstig für den Spieler. Anders ist das bei der betrachteten probabilistischen Strategie: Diese enthält regelmäßig günstige Teilsequenzen - etwa der Form 'AAB', 'BAABA' oder 'ABBAB'-, sodass sich insgesamt ein günstiges Spiel ergibt.

2.3 Optimale Strategien

Die Frage nach der optimalen probabilistischen Strategie lässt sich grob heuristisch beantworten, indem eine Simulation mit allen Werten im Intervall $[0, 1]$ durchgeführt wird. Freilich gibt es davon unendlich viele. Begnügt man sich zunächst mit zweistelliger Genauigkeit, ergibt sich der Graph aus Abbildung 2.

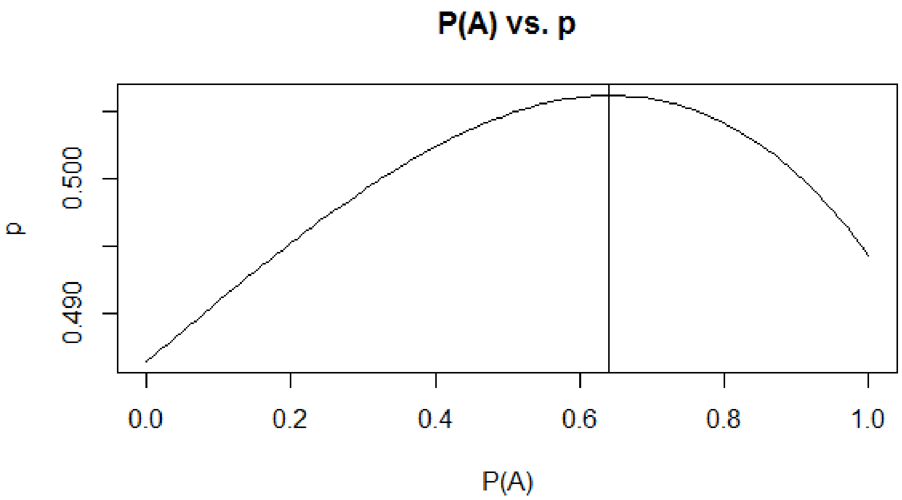

Abbildung 2: Leistungsfähigkeit probabilisitscher Strategien

Das Maximum liegt bei $P(A) = 0.64$, wobei dann die Wahrscheinlichkeit für einen Rundengewinn etwa $p = 0.5062$ beträgt. Eine exaktere Lösung liefert die Formulierung einer Zielfunktion, die unter Einhaltung von Nebenbedingungen maximiert wird[50]. Um die Wahrscheinlichkeit für einen Rundengewinn zu maximieren, sollte P(A) 0.643132 betragen. Langfristig kann der Spieler dann in 50.6232105% der Runden mit einem Gewinn rechnen.

Damit ist klar, dass es für den Spieler nicht optimal sein kann, eine probabilistische Strategie zu wählen, denn die bisher beste betrachtete deterministische Strategie 'AAB' liefert ein $p = 0.529552\overline{216}$ und ist damit jeder probabilistischen Strategie überlegen. Wird jede Sequenz als Binärzahl interpretiert (z.B. $AAB \triangleq 110$), so folgt aus der Unbeschränktheit der natürlichen Zahlen die Unbeschränktheit der Menge der Sequenzen mit endlicher Periodenlänge. Da das Intervall deterministischer Strategien mit endlicher Periodenlänge nicht beschränkt ist, ist die Suche nach der optimalen deterministischen Strategie komplizierter als zuletzt bei den probabilistischen Strategien. Einen ersten Anhaltspunkt liefert folgende Heuristik: Alle Zahlen im Intervall $[0, 1024)$ werden in Binärzahlen umgewandelt, sodass sie problemlos als Spielsequenz interpretiert werden können, und es wird eine Simulation mit allen solchen Sequenzen (bis einschließlich zur Periodenlänge 10) durchgeführt. Zu beachten ist dabei, dass im Limit etwa die Sequenz 'BBBBBBBBBA' potentiell[51] zum selben Ergebnis führt wie die Sequenz 'ABBBBBBBBB'. Während letztere Sequenz als Binärzahl im Intervall $[0, 1024)$ enthalten ist, kommt erstgenannte Sequenz nicht vor, da alle Bs vor dem ersten A „abgeschnitten“ werden. Trotzdem sind alle für die Grenzbetrachtung relevanten Sequenzen vorhanden, da das Muster - wie am obigen Beispiel gezeigt - dann um wenige Stellen verschoben in gleicher Weise vorkommt. Bei einer Milliarde Iterationen pro Sequenz im Rahmen der Simulation spielt dies keine Rolle. Es ergibt sich folgendes Bild:

[50]Siehe Anhang, 'Optimale probabilistische Strategie'.

[51]Ungenauigkeit ab der fünften Nachkommastelle besteht aufgrund der Tatsache, dass es sich um eine Monte-Carlo-Simulation handelt.

	row.names	dec	bin	p
1	694	693	1010110101	0.5336507
2	27	26	11010	0.5336442
3	23	22	10110	0.5336329
4	859	858	1101011010	0.5336298
5	22	21	10101	0.5336230
6	727	726	1011010110	0.5336105
7	87	86	1010110	0.5299871
8	91	90	1011010	0.5299793
9	86	85	1010101	0.5299726
10	107	106	1101010	0.5299347
11	55	54	110110	0.5295559
12	46	45	101101	0.5295550
13	366	365	101101101	0.5295516
14	6	5	101	0.5295446

Abbildung 3: Leistungsfähigste Sequenzen bis zur Länge 10 im Vergleich

Die sechs Sequenzen mit dem höchsten erwarteten Rundengewinn sind alle Vertauschungen der periodischen Anwendung der Sequenz 'BABAA'. Bis auf wenige Stellen am Anfang und Ende der Simulation, die bei einer Milliarde Iterationen zu vernachlässigen sind, entsprechen sie alle der Strategie, die Sequenz 'BABAA' periodisch zu spielen. 'BABAA' liefert durch Simulation etwa $p = 0.5336$. Das ist das Maximum aller Sequenzen bis zu einer Länge von 10 (inklusive). Obwohl die Vermutung, dass es sich bei 'BABAA' um die optimale Sequenz aus der unbeschränkten Menge endlicher Sequenzen handelt, aufgrund der bisher angestellten Überlegungen naheliegend ist, ist das hiermit noch nicht gezeigt. Im Folgenden wird die Idee eines auf Rückwärtsinduktion basierender Algorithmus skizziert, mit dessen Hilfe sich die Optimalität von 'BABAA' unter allen endlichen Sequenzen nachweisen lässt[52].

Die Übergangsmatrizen für die Spiele A und B sind:

$$\mathbf{P}_A = \begin{array}{c} \\ s_0 \\ s_1 \\ s_2 \end{array} \begin{array}{c} \begin{array}{ccc} s_0 & s_1 & s_2 \end{array} \\ \begin{pmatrix} 0 & \frac{3}{37} & \frac{34}{37} \\ \frac{9}{37} & 0 & \frac{28}{37} \\ \frac{28}{37} & \frac{9}{37} & 0 \end{pmatrix} \end{array}, \mathbf{P}_B = \begin{array}{c} \\ s_0 \\ s_1 \\ s_2 \end{array} \begin{array}{c} \begin{array}{ccc} s_0 & s_1 & s_2 \end{array} \\ \begin{pmatrix} 0 & \frac{18}{37} & \frac{19}{37} \\ \frac{19}{37} & 0 & \frac{18}{37} \\ \frac{18}{37} & \frac{19}{37} & 0 \end{pmatrix} \end{array}.$$

Die Wahrscheinlichkeit, Spiel A zum Zeitpunkt t zu gewinnen, beträgt $p_{winA}(t) = \pi_0(t)\frac{3}{37} + (1 - \pi_0(t))\frac{28}{37}$, wobei $\pi_0(t)$ der Wahrscheinlichkeit entspricht, sich zum Zeitpunkt t im Zustand s_0 zu befinden. Entsprechend bezeichnen $\pi_1(t)$ und $\pi_2(t)$ die Wahrscheinlichkeiten, sich zum Zeitpunkt t in Zustand s_1 bzw. s_2 zu befinden. Sei $\pi(t) =$

[52]Vgl. zum Algorithmus [Dinis, 2008, S. 2f]

$(\pi_0(t), \pi_1(t), \pi_2(t))^T$ mit $\pi_0(t) + \pi_1(t) + \pi_2(t) = 1$. Dann gilt $\pi(t+1) = \mathbf{P}_A^T \pi(t)$, falls Spiel A gespielt wird, und $\pi(t+1) = \mathbf{P}_B^T \pi(t)$, falls Spiel B gespielt wird. Sei der erwartete Gewinn $g(\pi_0(t)) = g^A(\pi_0(t))$, falls Spiel A gespielt wird, und $g(\pi_0(t)) = g^B$, falls Spiel B gespielt wird mit $g^A(\pi_0(t)) = 2[\pi_0(t)\frac{3}{37} + (1 - \pi_0(t))\frac{28}{37}] - 1$ und $g^B = 2\frac{18}{37} - 1$. Bezeichne $\tilde{G}_n(\pi)$ den maximierten erwarteten Gewinn unter der Wahrscheinlichkeitsverteilung π, wenn noch n weitere Runden zu spielen sind. Dann gilt $\tilde{G}_1(\pi) = \max\{g^A(\pi_0), g^B\}$, und, falls n > 1, $\tilde{G}_n(\pi) = \max\{g^A(\pi_0) + \tilde{G}_{n-1}(\mathbf{P}_A^T \pi), g^B + \tilde{G}_{n-1}(\mathbf{P}_B^T \pi)\}$.

Es werde nun bei n = 1 (eine letzte Runde wird gespielt) begonnen und angegeben, bis zu welchem π_0 die Entscheidung auf Spiel A fällt. Als nächstes werde die Runde n = 2 betrachtet: Da π_0 bei der letzten Entscheidung davon abhängt, ob sich der Spieler in der Runde zuvor für Spiel A oder B entschieden hat, muss nun (und in jedem folgenden "Rückwärtsschritt") jede mögliche Kombination aus π_0 und π_1 betrachtet werden, und für die betreffende Kombination angegeben werden, für welches Sub-Spiel (A oder B) man sich unter Berücksichtigung der Optimalitätsbedingung $\tilde{G}_n(\pi)$ entscheidet.[53] Dieser Vorgang wird solange wiederholt, bis n der Anzahl der insgesamt betrachteten Runden entspricht.[54] Auf diese Weise lässt sich zu einer beliebigen Anfangsverteilung die optimale Sequenz aus Spielen A und B bestimmen. Für Sequenzen hinreichender Länge ergibt sich - je nach gewählter Anfangsverteilung früher oder später - die periodische Entwicklung 'BABAA'. Eine einfache rekursive Implementierung des Algorithmus mit Sequenzen hinreichender Länge, die in annehmbarer Zeit eine Lösung liefert, scheitert allerdings an der exponentiellen Laufzeit. Deshalb sei an dieser Stelle auf den elaborierteren Ansatz verwiesen, den Dinis anstelle einfacher Rückwärtsinduktion verfolgt[55].

2.4 Die Kanonische Form des Spiels und der Unfairness-Parameter ϵ

Das Spiel des Parrondo-Paradoxons kann in einer allgemeineren Form beschrieben werden, sodass sich untersuchen lässt, wie sich das Ausmaß des Nachteils, das der Spieler bei den Subspielen gegenüber dem Casino hat (Unfairness-Parameter), auf das Verhalten des Gesamtsystems auswirkt[56]. Spiel A_ϵ^K liefert dem Spieler mit $p_z = \frac{1}{10} - \epsilon$ einen Gewinn, falls sein Vermögen durch 3 teilbar ist und mit $p_a = \frac{3}{4} - \epsilon$ einen Gewinn, falls das Vermögen kein Vielfaches von 3 ist. Andernfalls verliert der Spieler. Spiel B_ϵ^K werde mit $p_{B_\epsilon^K} = \frac{1}{2} - \epsilon$ gewonnen und mit entsprechender Gegenwahrscheinlichkeit verloren.

[53] Wegen $\pi_0 + \pi_1 + \pi_2 = 1$ ist das System durch den zweidimensionalen Vektor (π_0, π_1) vollständig determiniert.

[54] Es werden nur endlich viele Sequenzen endlicher Länge betrachtet. Daher ist gesichert, dass der Algorithmus in endlicher Zeit terminiert.

[55] Vgl. [Dinis, 2008, S. 3ff]

[56] Zu dieser sog. kanonischen Form des Spiels siehe etwa [Harmer and Abbott, 1999, S. 207ff]

Der Unfairness-Parameter ist ein kleines, nicht negatives $\epsilon \in \mathbb{R}$. Wie zuvor bekommt der Spieler eine Geldeinheit, falls er gewinnt, und verliert eine Geldeinheit andernfalls.

Sei $\epsilon = 0$. Dann ist

$$\mathbf{P}_{A_0^K} = \begin{array}{c c} & \begin{array}{ccc} s_0 & s_1 & s_2 \end{array} \\ \begin{array}{c} s_0 \\ s_1 \\ s_2 \end{array} & \begin{pmatrix} 0 & \frac{1}{10} & \frac{9}{10} \\ \frac{1}{4} & 0 & \frac{3}{4} \\ \frac{3}{4} & \frac{1}{4} & 0 \end{pmatrix} \end{array} \text{ und } \lim_{n\to\infty} \mathbf{P}^n_{A_0^K} = \begin{array}{c c} & \begin{array}{ccc} s_0 & s_1 & s_2 \end{array} \\ \begin{array}{c} s_0 \\ s_1 \\ s_2 \end{array} & \begin{pmatrix} \frac{5}{13} & \frac{2}{13} & \frac{6}{13} \\ \frac{5}{13} & \frac{2}{13} & \frac{6}{13} \\ \frac{5}{13} & \frac{2}{13} & \frac{6}{13} \end{pmatrix} \end{array}.$$

$p_{A_0^K} = \frac{5}{13} * \frac{1}{10} + \frac{8}{13} * \frac{3}{4} = \frac{1}{2}$. Beide Sub-Spiele werden für $\epsilon = 0$ als fair bezeichnet. Dass das Spiel B_0^K weder günstig noch ungünstig ist, ergibt sich aus folgendem Argument: Spiel B_0^K ist symmetrisch, da Casino und Spieler aufgrund der Unmöglichkeit eines erfolgreichen Glücksspielsystems auf Basis eines Bernoulli-Prozesses mit dem selben Erwartungswert und der selben Varianz konfrontiert sind. Außerdem ist B_0^K ein Nullsummenspiel. Ein symmetrisches Nullsummenspiel kann für keinen Spieler günstig oder ungünstig sein.[57]

Auch Spiel A_0^K ist weder günstig noch ungünstig. Denn wäre dies der Fall, müsste die Wahrscheinlichkeit, dass ein beliebig kleiner Gewinn bzw. Verlust, $S_n - n$, im Limit überschritten wird, gegen eins streben. Da jedoch im Limit aus $p_{A_0^K} = \frac{1}{2}$ folgt, dass $\forall n \in \mathbb{N} : P(S_n - n > 0) = P(S_n - n < 0) < \frac{1}{2}$, kann dieser Fall niemals eintreten.

Wie gezeigt werden kann, ist die Sequenz 'BABAA' auch für die kanonische Form des Spiels mit $\epsilon = 0$ optimal[58]. In diesem Fall gewinnt der Spieler in etwa 53.785% der Zeit. Die Entwicklung von p für $0 \leq \epsilon \leq 0.04$ unter Anwendung der Strategie, 'BABAA' periodisch zu spielen, zeigt folgender Graph:

[57] Wenn ein Zweipersonen-Spiel symmetrisch ist, muss es für beide günstig, ungünstig oder weder günstig noch ungünstig sein. Ein Nullsummenspiel kann aber nicht für alle beteiligten Spieler günstig bzw. ungünstig sein.

[58] Vgl. [Dinis, 2008, S. 4]

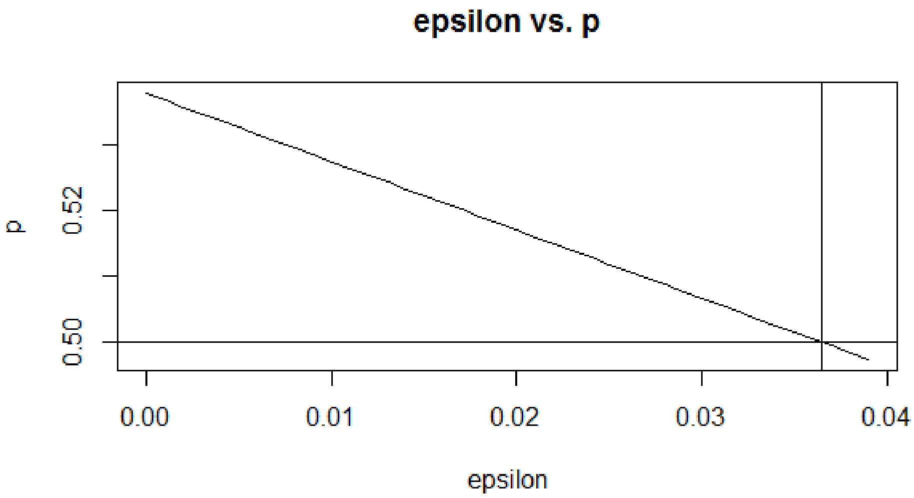

Abbildung 4: Leistungsfähigkeit von 'BABAA' vs. Unfairness-Parameter

Es ist also günstig, bis zu einem ϵ von etwa 0.0365, die Strategie 'BABAA' zu wählen. Der betrachtete Ausschnitt zeigt einen linearen Zusammenhang zwischen p und ϵ. Nun soll untersucht werden, ob für ein $\epsilon > 0.0365$ eine andere Sequenz gefunden werden kann, für die sich ein günstiges Spiel ergibt. Eine Simulation mit allen ein- bis zehnstelligen Sequenzen liefert für ein ϵ von 0.03655 kein p über $\frac{1}{2}$. Es kann gezeigt werden, dass dies auch für Sequenzen mit Periodenlänge größer 10 gilt[59].

Um das Verhalten probabilistischer Strategien unter Variation des Unfairness-Parameters ϵ zu untersuchen, wird die Übergangsmatrix $\mathbf{P}_{P_\epsilon^K} = P(A_\epsilon^K) * \mathbf{P}_{A_\epsilon^K} + (1 - P(A_\epsilon^K)) * \mathbf{P}_{B_\epsilon^K}$ gebildet. $\mathbf{P}_{P_\epsilon^K}$ hat folgende allgemeine Form:

$$\begin{pmatrix} 0 & P(A) * p_z + (1 - P(A)) * p_B & P(A) * (1 - p_z) + (1 - P(A)) * (1 - p_B) \\ P(A) * (1 - p_a) + (1 - P(A)) * (1 - p_a) & 0 & P(A) * p_a + (1 - P(A)) * p_B \\ P(A) * p_a + (1 - P(A)) * p_B & P(A) * (1 - p_a) + (1 - P(A)) * (1 - p_B) & 0 \end{pmatrix}$$

Damit lässt sich p für alle Kombinationen aus P(A) und ϵ bestimmen. Das maximale p von 0.5130994 wird bei einem P(A) von 0.59 erreicht, unter der Voraussetzung, dass ϵ gleich null ist. Bei einem ϵ von 0.0137 und einem P(A) von 0.62 beträgt p annähernd $\frac{1}{2}$. Für alle $\epsilon > 0.0137$ ist jede probabilistische Strategie für den Spieler ungünstig. Berechnungen von p über die stationäre Wahrscheinlichkeitsverteilung ergeben folgendes Bild:

[59] Vgl. [Dinis, 2008, S.3f]

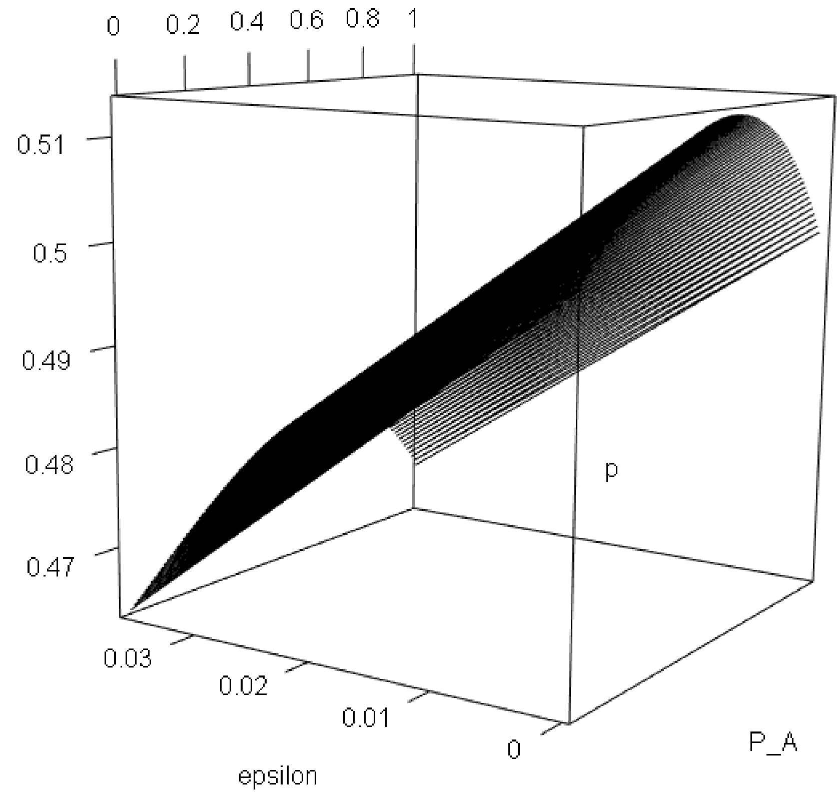

Abbildung 5: Probabilistische Strategien vs. Unfairness-Parameter

3 Parrondo-Paradoxon und Kooperation

3.1 Entstehung von Kooperation

Die Entstehung und Aufrechterhaltung von Kooperation zwischen Individuen in einem System ist Untersuchungsfeld unterschiedlichster Wissenschaften. Das gilt insbesondere auch für die Wirtschafts- und Sozialwissenschaften. Zur Entwicklung von Kooperation zwischen - mehr oder weniger - egoistisch eingestellten, ihren eigenen Nutzen maximierenden Individuen gibt es eine Vielzahl an Theorien und unzählige Modelle.

Ein richtungsweisendes (Computer-)Experiment wurde von Axelrod und Hamilton zu Beginn der 1980er durchgeführt und unter dem Titel „The Evolution of Cooperation“ sowohl als Paper als auch in Buchform veröffentlicht[60]. Bei diesem Experiment in Gestalt eines Tourniers traten verschiedenste mehr oder weniger ausgeklügelte Strategien in einem iterierten Dilemma-Spiel gegeneinander an. Es zeigte sich, dass „wohlwollende“ Strategien[61] den größten Erfolg brachten. Kooperation machte sich unter den gegebenen Bedingungen also bezahlt.

Der hier gewählte Ansatz geht einen etwas anderen Weg: Untersucht werden soll ein Netzwerk von Agenten, die ein gegebenes symmetrisches Spiel[62] wiederholt gegen ihre unmittelbaren Nachbarn spielen. Von Zeit zu Zeit bekommen die Agenten die Möglichkeit, ihre Strategie zu revidieren. Dabei orientieren sie sich an ihren erfolgreicheren Nachbarn. Es wird allerdings angenommen, dass der Imitationsprozess nicht immer einwandfrei funktioniert, und daher nicht immer der erfolgreichste Nachbar imitiert werden kann.

Mithilfe einer Computersimulation des iterierten Spiels im Netzwerk soll gezeigt werden, dass der Kooperationslevel - und damit die Gesamtwohlfahrt - innerhalb des Systems drastisch erhöht werden kann, sofern der Imitationsprozess nicht immer (aber auch nicht nie) perfekt funktioniert. Dieser Effekt weist einige Parallelen zum Parrondo Paradox auf: Während bei vollständig perfekter und vollständig imperfekter Imitation für eine Vielzahl von Spielen jegliche - oder zumindest ein Großteil der - Kooperation innerhalb des Systems verloren geht, wird oftmals ein relativ hoher Kooperationslevel erreicht, sobald perfekte und imperfekte Imitation stochastisch kombiniert werden. Da in Systemen ohne Kooperation oder mit niedrigem Kooperationslevel die Gesamtwohlfahrt als Summe

[60] [Axelrod and Hamilton, 1981]

[61] Diese Strategien versuchen nie, von sich aus den Mitspieler zu hintergehen. Außerdem vergeben sie schnell: Sobald der Mitspieler kooperiert, kooperieren sie auch sehr bald wieder.

[62] Sei S die Menge möglicher Strategien, die jedem Spieler zur Verfügung stehen. Dann ist ein symmetrisches n-Personen-Spiel eine Funktion $S^n \rightarrow \mathbb{R}^n$, die jedem n-dimensionalen Strategievektor einen n-dimensionalen Auszahlungsvektor zuweist.

der Auszahlungen der einzelnen Agenten im vorliegenden Kontext signifikant niedriger ist als in einem System mit hohem Kooperationsniveau, können vollständig perfekte und vollständig imperfekte Imitation für das System als Gesamtheit als Verluststrategien zu betrachtet werden.

3.2 Soziale Dilemmata

Spieltheorie untersucht Entscheidungsprobleme, in die mehrere Agenten involviert sind. Dabei beeinflussen die Entscheidungen aller teilnehmenden Agenten den Ausgang einer spieltheoretischen Problemsituation. Häufig ist dieser Ausgang nicht paretooptimal, obwohl die Agenten aus ihrer (isolierten) Sicht rational und damit gewinnmaximierend handeln. Es wird dann von einer sozialen Dilemmasituation gesprochen, da beteiligte Agenten besser gestellt werden könnten, ohne einen anderen Beteiligten schlechter stellen zu müssen. Liegt eine solche Situation vor, ist es von Interesse, die Rahmenbedingungen so zu ändern, dass die Agenten kooperieren und sich somit in Richtung des Paretooptimums bewegen.

Spieltheorie wird zur Modellierung von Problemstellungen in Geistes-, Sozial-, Ingenieurs- und Naturwissenschaften verwendet, weshalb es sich bei Agenten im Allgemeinen um verschiedenste Gebilde, wie bspw. Staaten(bünde), Personen(gruppen), Tiere, Einzeller, Viren, Computeralgorithmen etc., handeln kann.

Eines der ältesten bekannten sozialen Dilemmata, das großen Einfluss auf die Entwicklung der ökonomischen Lehre hatte, und regelmäßig mithilfe der Spieltheorie modelliert wird, ist die Tragik der Allmende[63]. Hierbei kommt es zu einer Übernutzung begrenzter gemeinschaftlich genutzter Ressourcen zum Nachteil aller Beteiligten, da Regeln zur Nutzung des Allgemeinguts fehlen (aktuelle Probleme: Treibhausgasausstoß in die gemeinsame Atmosphäre, Überfischung der Weltmeere). Ohne Kooperation und die Errichtung gemeinsamer bindender Regeln kommt es dann häufig zu einem 'Race to the Bottom', der bis hin zur Vernichtung lebensgrundlegender Ressourcen führen kann (z.B. Fällung des letzten Baums durch Ureinwohner der Osterinsel).

Im Folgenden werden symmetrische Zwei-Personen-Spiele betrachtet, deren bekanntester Vertreter das Gefangenendilemmaspiel ist. Bei diesen Spielen handelt es sich um Nicht-Nullsummen-Spiele - der Gewinn des einen muss folglich nicht dem Verlust des anderen entsprechen. In den hier betrachteten Spielen hat jeder der zwei Spieler die Strategien 'kooperieren' oder 'betrügen' zur Auswahl. Es gibt also vier Möglichkeiten, wie eine Spielrunde enden kann: Beide Spieler kooperieren, Spieler 1 kooperiert und

[63] [Hardin, 1968]

Spieler 2 betrügt, Spieler 1 betrügt und Spieler 2 kooperiert, beide Spieler betrügen. Allgemein sieht die Auszahlungsmatrix für den Zeilenspieler folgendermaßen aus[64]

$$\begin{array}{cc} & \begin{array}{cc} C & D \end{array} \\ \begin{array}{c} C \\ D \end{array} & \begin{pmatrix} 1 & S \\ T & 0 \end{pmatrix}. \end{array}$$

S ('sucker' bzw. 'Trottel') bezeichnet die Auszahlung für den Spieler, der naiv genug ist, zu kooperieren, während der andere Spieler betrügt. T ('temptation' bzw. 'Verlockung') ist die Auszahlung, die ein Spieler erhält, wenn es ihm gelingt zu betrügen, während der andere Spieler kooperiert. Für das Gefangenendilemma gilt: $-1 < S < 0$ und $1 < T < 2$. Das Taube-Falke-Spiel wird allgemein durch $0 < S < 1 < T < 2$ beschrieben[65]. Beim Gefangenendilemma haben beide Spieler die strikt dominante Strategie D[66], sodass ein Nashgleichgewicht mit der Auszahlung von 0 für beide Spieler vorliegt. Das Taube-Falke-Spiel besitzt keine dominante Strategie und zwei Nashgleichewichte in reinen Strategien, bei denen jeweils ein Spieler kooperiert (Auszahlung S) und der andere betrügt (Auszahlung T).

Während beim Taube-Falke-Spiel keine Pareto-Verbesserung in einem Nash-Gleichgewicht in reinen Strategien möglich ist (der betrügende Spieler würde durch Kooperation schlechter gestellt werden), ist beim Gefangenendilemma eine Besserstellung für die Agenten möglich: Wenn beide Spieler kooperieren anstatt zu betrügen, könnten sie beide eine Auszahlung von 1 erreichen. Dies stellt im Vergleich zum Nash-Gleichgewicht, bei dem sich beide Spieler mit einer Auszahlung von 0 zufriedengeben müssen, eine Pareto-Verbesserung dar. Tatsächlich handelt es sich bei der Situation, in der beide Spieler im Gefangenendilemma kooperieren, um das Gesamtwohlfahrtsoptimum, da die Summe der Auszahlungen der beiden Spieler den Wert 2 nicht übersteigen kann.

3.3 Netzwerke

Netzwerke sind Systeme, die aus einer Menge von Elementen und Verbindungen zwischen diesen Elementen bestehen. Die mathematische Graphentheorie ist geeignet, Netzwerke als Spezialfall einer Relationen zu modellieren. Dabei entsprechen die Netzwerkelemente den Knoten und die Verbindungen zwischen ihnen den Kanten eines Graphen. Die im Folgenden betrachteten Graphen sind ungerichtet, ungewichtet und schlicht. Eine Kante

[64] Da es sich um symmetrische Spiele handelt, sind Zeilen- und Spaltenspieler vertauschbar.
[65] Vgl. [Roca et al., 2009, S. 1f]
[66] Eine dominante Strategie zeichnet sich dadurch aus, dass sie unabhängig vom Verhalten des Mitspielers, zum bestmöglichen Ergebnis führt. Im Gefangenendilemma führt die Wahl von Strategie D unabhängig vom Verhalten des Mitspielers immer zu einer höheren Auszahlung als Strategie C.

zwischen Elementen a und b zeigt an, dass eine Nachbarschaftsbeziehung zwischen den beiden Elementen besteht, wobei diese Beziehung im gegebenen Kontext symmetrisch, irreflexiv und nicht transitiv ist. Die Anzahl der Nachbarn eines Knotens wird als dessen Knotengrad bezeichnet. Die Entfernung zwischen zwei benachbarten Knoten wird im gegebenen Kontext nicht bewertet, da es lediglich darauf ankommen soll, ob zwei Elemente benachbart sind oder nicht.

Betrachtet werden zwei Arten von Netzwerken: gitterförmige[67] sowie skalenfreie bzw. -invariante Netzwerke[68]. Bei ersteren sind die Elemente - bildlich gesprochen - regelmäßig, entsprechend eines Schachbrettmusters angeordnet. Hier bilden die jeweils acht umliegenden Elemente die Nachbarschaft eines Knotens, wobei die Randbedingung periodisch gewählt wird[69]. Der Aufbau dieser Netzwerke ist vollkommen regelmäßig (jedes Element hat genau acht Nachbarn) und wirkt daher für Anwendungen der evolutionären Spieltheorie in einem sozialwissenschaftlichen Kontext relatitätsfern. Die Stärke dieser Art von Netzwerken liegt allerdings in der guten Visualisierbarkeit von Netzwerkprozessen.

In vielen sozialwissenschaftlichen Anwendungsbereichen (aber nicht nur dort) stellen skalenfreie Netzwerke eine gute Annäherung an die Realität dar[70]. Diese Netzwerke zeichnen sich dadurch aus, dass Nachbarschaften nicht gleichmäßig über die Netzwerkelemente verteilt sind, sondern in ihrer Verteilung einem Potenzgesetz folgen. Das Barabási-Albert-Modell stellt einen Algorithmus zur Bildung eines solchen skalenfreien Netzes zur Verfügung[71]:

- Konstruiere ein beliebiges Netzwerk mit m_0 zusammenhängenden Knoten
- Füge der Reihe nach neue Knoten hinzu
  - Jeder neue Knoten wird mit $m \leq m_0$ verschiedenen Knoten des Netzwerks verbunden
  - Die Wahrscheinlichkeit, dass ein neu hinzuzufügender Knoten mit dem sich bereits im Netzwerk befindenden Knoten i verbunden wird, beträgt:
    $p_i = \frac{k_i}{\sum_j k_j}$

[67]Engl.: Lattice Network
[68]Engl.: Scale-free Network
[69]Der rechte Nachbar eines Knotens am rechten Rand ist der Knoten am linken Rand der selben Zeile. Der obere Nachbar eines Knotens am oberen Rand ist der Knoten am unteren Rand der selben Spalte usw. Der rechte obere Nachbar eines Knotens an der rechten oberen Ecke ist der Knoten in der linken unteren Ecke usw.
[70]Vgl. [Barabási and Albert, 2002, S. 48]
[71]Vgl. [Barabási and Albert, 2002, S. 71]

– k_i bezeichnet den Knotengrad des Knotens i und $\sum_j k_j$ die Summe aller Knotengrade von Knoten im Netzwerk - entspricht also der doppelten Kardinalität der Kantenmenge

Das so entstehende Netzwerk besitzt i.d.R. einige wenige Knoten mit einem sehr hohen Knotengrad ('hubs' bzw. 'Netzwerkknotenpunkte'). Die Anzahl von Nachbarn dieser Netzwerkknotenpunkte liegt typischwersie weit über m, während der Großteil der verbleibenden Knoten nicht (bedeutend) mehr als m Nachbarn hat. In der Praxis stellen etwa Global Players, Personen mit sehr vielen Kontakten in sozialen Netzwerken oder Staaten mit außergewöhnlich vielen wirtschaftlichen Verflechtungen ins Ausland solche Knotenpunkte dar, die eine zentrale Bedeutung für die Dynamik innerhalb des skalenfreien Netzwerks einnehmen.

Lange Zeit wurden reine Zufallsgraphen zur Modellierung sozialer Netzwerke verwendet[72], da mit ihrer Hilfe Kleine-Welt-Phänomen nachgebildet werden kann: Auch in sehr großen zusammenhängenden Netzwerken beinhaltet der kürzeste Weg zwischen zwei Knoten i.d.R. verhältnismäßig wenige Kanten. Allerdings wurde - etwa durch Untersuchungen des WWW in den späten 1990ern - entdeckt, dass Zufallsgraphen eine deutlich geringere Clusterbildung aufweisen als jene Netzwerke, die mit ihrer Hilfe modelliert werden sollten. Skalenfreie Netzwerke lösen dieses Problem, da sie ebenfalls das Kleine-Welt-Phänomen aufweisen, aber im Gegensatz zu reinen Zufallsgraphen ein Clustering-Verhalten ähnlich dem vieler natürlich gewachsener Netzwerke besitzen[73].

Im Simulationsmodell werden u.a. quadratische gitterförmige Netzwerke mit achter-Nachbarschaft betrachtet, sodass für die Konstruktion dieser Netzwerke lediglich ein Größenparameter (als Quadratzahl) gegeben sein muss. Für die Konstruktion eines skalenfreien Netzwerks muss dessen Größe (als beliebige natürliche Zahl größer dem durchschnittlichen Knotengrad) sowie der durchschnittliche Knotengrad gegeben sein. Zudem muss für beide Netzwerkarten festgelegt werden, welcher Anteil der Agenten zu Beginn der Simulation eine kooperative Strategie besitzen soll[74].

3.4 Imitationsregeln

Imitation ist ein wesentliches Merkmal des menschlichen Sozialverhaltens. Dabei ist es naheliegend anzunehmend, dass jene Individuen, deren Verhalten von anderen als erfolgreich bewertet wird, mit größerer Wahrscheinlichkeit imitiert werden als ihre weniger

[72] Vgl. etwa [Barabási and Albert, 2002, S. 54]

[73] Vgl. [Barabási and Albert, 2002, S. 75f]

[74] Die initiale Zuteilung von Strategien erfolgt bei der Konstruktion des Netzwerks zufällig mit Wahrscheinlichkeitsparameter P(COOP)

erfolgreichen Zeitgenossen. Allerdings ist es einem Individuum innerhalb eines sozialen Netzwerks kaum möglich, das Verhalten aller anderen Individuen im Auge zu behalten. Daher wird angenommen, dass ausschließlich Nachbarn eines Individuums imitiert werden können.

Zwei Formen der Imitation werden im Folgenden unterschieden[75]:

- 'Unconditional Imitation Rule' bzw. UIR: Der erfolgreichste Nachbar wird imitiert, sofern er erfolgreicher ist als das Individuum, das seine Strategie revidiert

- 'Replicator Rule' bzw. RR: Ein zufällig gewählter Nachbar wird mit einer Wahrscheinlichkeit, die proportional zu dessen relativem Erfolg[76] ist, imitiert, falls er erfolgreicher ist als das Individuum, das seine Strategie revidiert

Die RR trägt dem Umstand Rechnung, dass es den Agenten nicht immer möglich ist, die gesamte Nachbarschaft im Auge zu behalten bzw. erfolgreiches Verhalten anderer eindeutig identifizieren und imitieren zu können. Auch in realen Situationen sind solche Fehler beim Informationsfluss bzw. bei der Informationsverarbeitung zumeist unvermeidlich.

Für das vorliegende Simulationsmodell bedeutet das: Jeder Agent des Netzwerks spielt gegen alle seine Nachbarn eine Runde des vorgegebenen Spiels, das durch die Parameter S und T festgelegt wird (Iteration). Die so erhaltetenen Auszahlungen einer Iteration werden für jeden Agenten aufsummiert und messen dessen absoluten Erfolg. Haben alle Agenten gegen alle ihre Nachbarn gespielt, und die Iteration ist damit beendet, revidieren sie simultan ihre jeweilige Strategie gemäß einer der oben gegebenen Imitationsregeln. Eine der beiden Regeln wird jedes mal aufs Neue für jeden Agenten, der dabei ist, seine Strategie zu revidieren, zufällig mit Wahrscheinlichkeitsparameter $P(UIR) = \rho$ bzw. $P(RR) = 1-\rho$ gewählt. Sobald alle Agenten ihre Strategie revidiert haben, wird ihr Erfolg auf null gesetzt, da beim Anpassen der Strategien nur die unmittelbar vorangegangene Iteration berücksichtigt werden soll.

3.5 Simulationsergebnisse und Interpretation

Zunächst wird das Gefangenendilemmaspiel betrachtet. Die Netzwerkgröße beträgt bei beiden Netzwerkarten N = 10 000. Für das skalenfreie Netzwerk wird der durchschnittliche Knotengrad auf etwa 8 gesetzt (m = 4), da dies auch der Anzahl der Nachbarn im gitterförmigen Netzwerk entspricht. P(COOP), also die Wahrscheinlichkeit, dass ein Agent bei der Initialisierung des Netzwerks eine kooperative Strategie besitzt, beträgt

[75] Vgl. zu den Imitationsregeln [Roca et al., 2009, S. 1]

[76] Differenz zwischen dem Erfolg des zufällig gewählten Nachbarn und des strategierevidierenden Individuums

0.5. Im Gitterförmigen Netzwerk beträgt die Sucker-Auszahlung $S_L = -0.7$, im skalenfreien Netzwerk $S_S = -0.3$. T, die Temptation-Auszahlung, wird in beiden Netzwerken auf 1.1 gesetzt. Es ergibt sich nach jeweils 10 000 Iterationen das folgende Bild unter Variation von ρ im Intervall $[0, 1]$ in Schritten der Größe 0.1:

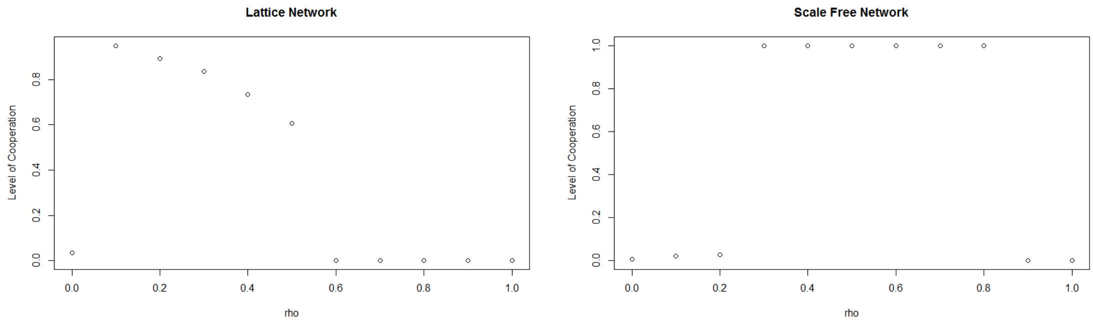

Abbildung 6: Kooperationsniveau in Abhängigkeit von rho, Gefangenendilemma

In beiden Fällen kann die stochastische Kombination zweier Imitationsstrategien bzw. -regeln, die isoliert angewandt zum unerwünschten Ausgang - gar keine bzw. kaum Kooperation - führen, dazu dienen, einen erwünschten Zustand - hohe Kooperation - herzustellen. Mithilfe des gitterförmigen Netzwerks lassen sich Dynamiken bei der Entstehung von Kooperation visualisieren[77]. Es formen sich beim Gefangenendilemmaspiel i.d.R. zu Beginn einige vereinzelte Cluster aus kooperierenden Agenten[78]. Zu diesem Zeitpunkt liegt der Anteil an kooperierenden Individuen im Netzwerk typischerweise zwischen 1 und 2 Prozent. Diese Cluster wachsen jedoch für bestimmte Werte von ρ im Verlauf der Simulation an, sodass sich dann ein Kooperationsniveau zwischen 60 und 100 Prozent erreichen lässt.

Es existiere zu Beginn bereits ein Cluster aus kooperierenden Agenten in einem gitterförmigen Netzwerk, bei dem ρ zunächst auf 0 gesetzt ist, also immer[79] die UIR zum Einsatz kommt. In diesem Fall glätten sich die Ränder des Clusters sehr schnell. In Tabelle 1 ist dies ersichtlich: Das Cluster aus dunkel eingefärbten kooperierenden Agenten besitzt nach wenigen Iterationen vollkommen glatte Ränder. Dann können die Cluster

[77]Siehe Anhang, 'Clusterbildung im Netzwerk'.
[78]Im folgenden werden Cluster aus kooperierenden Agenten auch kurz mit 'Cluster' bezeichnet.
[79]Die zur Generierung von Pseudozufallszahlen verwendete Methode java.lang.Math.random() liefert mit Wahrscheinlichkeit $\frac{1}{10^{16}}$ den Wert 0.

kooperiernder Agenten offensichtlich nicht weiter anwachsen, da die Agenten an den Rändern der Cluster weniger erfolgreich sind als ihre nichtkooperierenden Nachbarn. Dieser Lock-In-Effekt führt dazu, dass das Kooperationsniveau für $\rho = 0$ beim Gefangenendilemmaspiel in einem Gitternetzwerk typischerweise zwischen 0 und 2 Prozent stagniert.

Tabelle 1: Entw. Clusterränder unter der UIR beim Gefangenendilemma

8.0	4.6	4.4	1.1		8.0	8.0	2.9	3.3
8.0	6.3	1.2	2.2		8.0	8.0	2.9	3.3
8.0	6.3	1.2	2.2	→	8.0	8.0	2.9	3.3
6.3	2.9	3.3	1.1		8.0	8.0	2.9	3.3
6.3	5.5	2.2	0.0		8.0	8.0	2.9	3.3
6.3	1.2	2.2	0.0		8.0	8.0	2.9	3.3

Falls der Wert für ρ auf 1 gesetzt ist (oder zumindest nahe 1 liegt) ist andererseits eine Invasion der Cluster durch nicht kooperierende Strategien wahrscheinlich und führt mit der Zeit zu Erosionserscheinungen an den Clusterrändern, wie es in Tabelle 2 dargestellt wird. Dadurch werden die kooperierenden Cluster instabil und können von der betrügerischen Strategie unterwandert werden. Im schlimmsten Fall führt dies zur Auflösung eines Clusters.

Tabelle 2: Mgl. Entw. Clusterränder unter der RR beim Gefangenendilemma

8.0	4.6	4.4	1.1		8.0	2.9	3.3	0.0		4.6	4.4	1.1	0.0
8.0	6.3	1.2	2.2		8.0	4.6	4.4	1.1		1.2	2.2	0.0	0.0
8.0	6.3	1.2	2.2	→	4.6	1.2	-2.2	1.1	→	4.4	1.1	0.0	0.0
6.3	2.9	3.3	1.1		7.7	6.6	4.4	2.2		5.5	0.0	0.0	0.0
6.3	5.5	2.2	0.0		4.6	-0.5	-3.9	1.1		5.5	0.0	0.0	0.0
6.3	1.2	2.2	0.0		6.3	6.6	3.3	1.1		3.3	2.2	1.1	0.0

Wenn beide Imitationsregeln kombiniert gespielt werden, ist die Gefahr einer Invasion durch die betrügerische Strategie geringer, da die höchste Auszahlung dann von den Agenten im Clusterinneren erreicht wird, und diese unter der UIR, sollte diese gewählt werden, sicher imitiert werden. Dieser Umstand stabilisiert die Clusterränder, sodass diese nicht zu sehr „ausfransen“. Eine beginnende Auflösung der Cluster kann somit durch regelmäßige Anwendung der UIR abgefangen und evtl. - zumindest zum Teil - rückgängig gemacht werden. Andererseits ist durch die stochastische Kombination der

Imitationsregeln auch gewährleistet, dass die Ränder nicht vollkommen flach werden. Das Clusterwachstum wird somit nicht frühzeitig zum erliegen kommen. In diesem Fall geht eine destabilisierende Dynamik, die das Cluster zwar angreifbar macht, jedoch dessen Wachstum erst ermöglicht, mit einer stabilisierenden Dynamik, die das Cluster gegen Angriffe von außen verteidigt, Hand in Hand.

Im Taube-Falke-Spiel kann, anders als im Gefangenendilemma-Spiel, durch gegenseitige Kooperation im Vergleich zu den Nash-Gleichgewichten keine Paretoverbesserung im Zwei-Personen-Spiel erreicht werden. Obwohl hier im zugrundeliegenden Spiel mit zwei Personen keine soziale Dilemmasituation vorliegt, kann, wie aus Tabelle 3 ersichtlich ist, die Entstehung von Kooperationsclustern zur Verbesserung der Gesamtwohlfahrt beitragen und im Extremfall sogar zu einer Paretoverbesserung führen. Auch beim Taube-Falke-Spiel lässt sich im gitterförmigen Netzwerk der Kooperationslevel für manche Spiel-Parameter durch Kombination der beiden Imitationsregeln erhöhen.

Tabelle 3: Wohlfahrtsvorteil durch Clusterbildung beim Taube-Falke-Spiel

6.8	5.2	6.8	5.2		8.0	8.0	8.0	8.0
5.2	6.8	5.2	6.8		8.0	8.0	8.0	8.0
6.8	5.2	6.8	5.2	$\rightarrow \cdots \rightarrow$	8.0	8.0	8.0	8.0
5.2	6.8	5.2	6.8		8.0	8.0	8.0	8.0
6.8	5.2	6.8	5.2		8.0	8.0	8.0	8.0
5.2	6.8	5.2	6.8		8.0	8.0	8.0	8.0

Im Taube-Falke-Spiel wird die Sucker-Auszahlung auf 0.7 gesetzt, die Temptation-Auszahlung auf 1.3. Die restlichen Parameter sind gleich wie zuvor in der Simulation des Gefangenendilemmas im Gitternetzwerk. Hier nimmt durch die Wahl von ρ zwischen 0.7 und 0.9 der Kooperationslevel deutlich zu. Allerdings sei angemerkt, dass das nicht immer gelten muss. Für andere Parameter und Netzwerktypen können reine Imitationsregeln zu einer höheren Effizienz führen als ihre stochastisch kombinierte Anwendung.

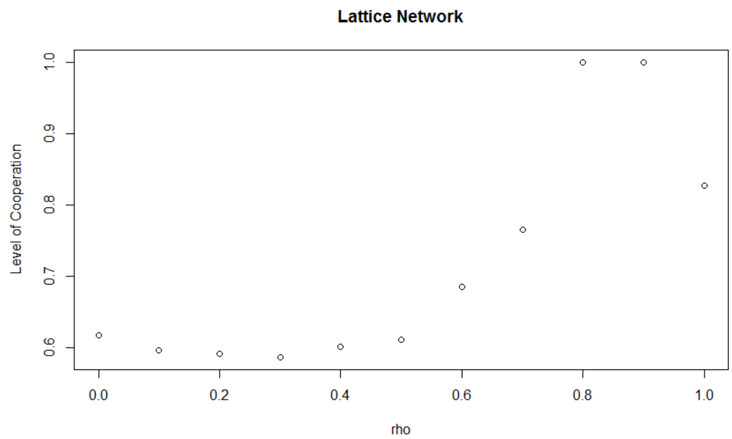

Abbildung 7: Kooperationsniveau in Abhängigkeit von ρ, Taube-Falke-Spiel

3.6 Diskussion und Conclusio

Wie sowohl für das Casino-Modell als auch für das Netzwerk-Modell gezeigt wurde, kann die Kombination zweier ineffizienter Strategien unter bestimmten Voraussetzungen zu einem effizienten Ergebnis führen. In beiden Fällen wurde eine lineare Dynamik mit einer nicht-linearen Dynamik kombiniert, wobei die Dynamiken stochastisch nicht unabhängig sind.

Spiel B kann als lineare Dynamik aufgefasst werden, da der Erwartungswert des Gesamtgewinns in Abhängigkeit der gespielten Runden n einen linear fallenden Verlauf aufweist. Dies ist bei Spiel A nicht der Fall; in Abhängigkeit des momentanen Spielkapitals entwickelt sich der Erwartungswert des Gesamtgewinns über die Zeit einem nicht-linearen Funktionsverlauf folgend[80].

Die ausschließliche Anwendung der UIR im Netzwerk-Modell führt bereits nach wenigen Runden zu vollkommen glatten Cluster-Rändern, die als lineare Struktur aufgefasst werden können. Damit stagniert das Wachstum von Kooperationsclustern i.d.R. bereits in einem frühen Stadium. Diese lineare Dynamik wird im untersuchten Fall mit der nicht-linearen RR kombiniert, sodass ein Stillstand des Cluster-Wachstums vermieden wird, ohne zu große Unruhe in die Kooperationscluster zu bringen, die schlussendlich eine Auflösung derselben bewirken würde.

Die stochastische Abhängigkeit der beiden Dynamiken führt in beiden Fällen der Kombination zu einer neuen Wahrscheinlichkeitsverteilung. Ob diese neue Wahrscheinlich-

[80] Vgl. hierzu [Harmer and Abbott, 1999, S. 209, Fig. 3]. Bezeichnung für Spiel A und B sind hier vertauscht.

keitsverteilung vorteilhaft ist, und die Effizienz des Gesamtsystems gesteigert werden kann, hängt von diversen Gegebenheiten und Parametern ab, wie etwa dem Unfairness-Parameter bzw. der gespielten Sequenz aus Spielen A und B im Casino-Modell oder der Netzwerkart bzw. der Höhe der Auszahlungen im Netzwerk-Modell. Kombinationen von Strategien, die unter bestimmten Gegebenheiten vorteilhaft sind, können daher unter anderen Gegebenheiten nachteilig gegenüber der unkombinierten Anwendung reiner Strategien[81] sein. Daraus ergibt sich, dass zwei Dynamiken mit ungünstigen Eigenschaften nur unter ganz bestimmten Voraussetzungen zu einer neuen günstigen kombiniert werden können, und keineswegs geschlossen werden darf, dass dies immer möglich ist. Diese Voraussetzungen zu finden und zu untersuchen, war Teil der vorliegenden Arbeit.

Ob parrondo-basierte Handelsstrategien auf realen Finanzmärkten Erfolg bringen können, ist umstritten. Zwar konnte ein gewisser Erfolg solcher Strategien in einem künstlichen Modell, in dem ein Netzwerk von Händlern simuliert wurde, nachgewiesen werden. Doch unterliegt das Ergebnis zahlreichen künstlichen Annahmen und Vereinfachungen, sodass die Anwendbarkeit auf realen Märkten offen bleibt und, wenn überhaupt, nur in speziellen Marktsituationen gegeben sein kann[82].

Ähnlich verhält es sich mit der Untersuchung von Kooperation in einem Netzwerk von Agenten, die ihre Nachbarn imitieren; auch hier wurden viele vereinfachende Annahmen getroffen, sodass Implikationen auf reale Netzwerke nur schwer abschätzbar sind. Das Gefangenendilemma bietet eine gute Basis zur Modellierung zahlreicher sozialer Dilemmata. Trotzdem kann es nur als Annäherung an reale Konflikte gesehen werden. Dass die Ergebnisse der Untersuchung eine gewisse Allgemeingültigkeit besitzen und weitgehend unabhängig vom Abstraktionsniveau des zugrundeliegenden Modells sind, zeugt schließlich vom potentiellen Einfluss parrondo-basierter Dynamiken auf die evolutionäre Spieltheorie[83].

[81] Nur Spiel A bzw. nur Spiel B oder $\rho \in \{0, 1\}$.
[82] Vgl. hierzu die Conclusio von [Boman et al., 2002]
[83] Vgl. [Roca et al., 2009, S. 4]

Literatur

R. Axelrod and W. D. Hamilton. The evolution of cooperation. Science, 211(4489): 1390–1396, 1981.

Albert-László Barabási and Réka Albert. Statistical mechanics of complex networks. Rev. Mod. Phys., 74:47–97, Jan 2002.

Magnus Boman, Stefan J. Johansson, and David Lybäck. Parrondo strategies for artificial traders. CoRR, cs.CE/0204051, 2002.

Luis Dinis. Optimal sequence for parrondo games. Phys. Rev. E, 77:021124, Feb 2008.

Rick Durrett. Probability: Theory and Examples. Cambridge University Press, 4 edition, 2010.

A. W. F. Edwards. Pascal's problem: The 'gambler's ruin. International Statistical Review / Revue Internationale de Statistique, 51(1):73–79, April 1983.

William Feller. An Introduction to Probability Theory and Its Applications, volume 1. Wiley, January 1968.

Garrett Hardin. The tragedy of the commons. Science, New Series, 162(3859):1243–1248, December 1968.

G. P. Harmer and D. Abbott. Parrondo's paradox. Statistical Science, 14(2):206–213, May 1999.

J. G. Kemeny and J. L. Snell. Finite Markov Chains. Springer, New York, 1976.

Carlos P Roca, José A Cuesta, and Angel Sánchez. Imperfect imitation can enhance cooperation. EPL (Europhysics Letters), 87(4):48005, 2009.

Mark Schilling. Pondering parrondo's paradox. Math Horizons, 16(1):12–13, 2008.

Appendices

Simulation des Casino-Modells

Sprache: Java

Verwendete Bibliotheken: java.util.*

```
// Interface fuer dynamisches Binden
public interface Game {

        boolean gamble();

}
```

```
public class GameA implements Game {

        private long x;

        public GameA(long x) {
                this.x = x;
        }

        @Override
        public boolean gamble() {
                int rndm = (int)(Math.random() * 37);
                return (x % 3 == 0 && rndm < 3) ||
                                        (x % 3 != 0 && rndm < 28);
        }
}
```

```
public class GameB implements Game {

        @Override
        public boolean gamble() {
                return (int)(Math.random() * 37) < 18;
        }
}
```

```java
import java.util.Scanner;

public class ParrondoSimulation {

 public static void main(String[] args) {
  // (Anfangs-)Vermoegen (mod 3)
  int x = (int)(Math.random() * 3);
  long counter = 0;      // Spielzug-Zaehler
  long won = 0;          // zaehlt gewonnene Spiele
  // Wahrscheinlichkeit fuer Spiel A
  double threshold = .5d;
  boolean detFlag = false; // deterministische Strategie?
  String sequence = ""; // Sequenz von Sub-Spielen
  Scanner sc = new Scanner(System.in); // Konsoleneingabe
  loop: while(true) {
   System.out.println(
    "Wahrscheinlichkeit fuer Spiel A oder Sequenz eingeben: ");
   if(sc.hasNextDouble()) {
    threshold = sc.nextDouble();
    if(threshold < 0 || threshold > 1) {
     System.out.println(
      "Wahrscheinlichkeit muss zwischen 0 und 1 liegen.");
     continue loop;
    }
   } else {
    sequence = sc.next();
    detFlag = true;
    for(int i = 0; i < sequence.length(); i++) {
     if(sequence.charAt(i) != 'A' &&
      sequence.charAt(i) != 'B') {
       System.out.println(
        "Sequenz darf nur aus A und B bestehen.");
       continue loop;
     }
    }
```

```
    }
    break loop;
   }
   while(counter < 1000000000) { // 1 Mrd Iterationen
    Game game = null;
    // falls deterministische Strategie:
    if(detFlag) {
     if(sequence.charAt((int)counter%sequence.length()) == 'A')
      game = new GameA(x);
     else
      game = new GameB();
    }
    // sonst: zufaellig mit entsprechender Wahrsceheinlichkeit
    // entscheiden, welches der beiden Spiele
    // durchgefuehrt wird
    else if(Math.random() < threshold) {
     game = new GameA(x);
    } else {
     game = new GameB();
    }
    // durchfuehren des Spiels; wenn gewonnen: Gewinnzaehler
    // und Vermoegen erhoehen, sonst Vermoegen senken
    if(game.gamble()) {
     won++;
     x = ++x % 3;
    } else {
     x = x == 0 ? 2 : x - 1;
    }
    // Spielzug-Zaehler erhoehen
    counter++;
   }
   // Anteil gewonnener Spiele ausgeben
   System.out.println((double)won / counter);
   sc.close();
  }
}
```

Berechnung der stationären Wahrscheinlichkeitsverteilung

Sprache: Java

Verwendete Bibliotheken: Jama (http://math.nist.gov/javanumerics/jama/) und java.util.*

Berechnung der stationären Wahrscheinlichkeitsverteilung:

http://introcs.cs.princeton.edu/java/95linear/MarkovChain.java.html (9.2.2015)

```
import java.util.Scanner;
import Jama.Matrix;

public class CalcStationaryMatrix {

 public static void main(String[] args) {

  // is the transition matrix cyclic?
  boolean cyclicFlag = false;
  Scanner sc = new Scanner(System.in);
  System.out.println("Enter dimension n of "
    + "nxn-matrix:");
  int N = sc.nextInt(); // dimension
  double[][] transition = new double[N][N];
  System.out.println("Enter \"c\" if matrix "
   + "is cyclic or anything else if not.");
  if(sc.next().equals("c"))
   cyclicFlag = true;
  System.out.println("Enter matrix:");
  // read matrix from console
  for(int i = 0; i < N; i++) {
   for(int j = 0; j < N; j++) {
    if(sc.hasNextDouble()) {
     transition[i][j] = sc.nextDouble();
    // if input is no double, check if it can
    // be interpreted as fraction
    } else {
     String frac = sc.next();
     String[] numDen = frac.split("/");
     try {
```

```
        if(numDen.length != 2)
         throw new Exception();
        // transform fraction to double
        transition[i][j] =
         (double)Integer.parseInt(numDen[0]) /
         Integer.parseInt(numDen[1]);
      } catch(Exception e) {
       System.out.println(
         "Floating-point number or
        + "simple fraction. End.");
       sc.close();
       return;
      }
     }
    }
   }
   sc.close();

   Matrix A = new Matrix(transition).transpose();
   Matrix x = null;
   if(cyclicFlag) {
    // If ergodic, stationary distribution =
    // unique solution to Ax = x
    // up to scaling factor.
    // We solve (A - I) x = 0, but replace
    // row 0 with constraint that
    // says the sum of x coordinates equals one
    Matrix B = A.minus(Matrix.identity(N, N));
    for (int j = 0; j < N; j++)
     B.set(0, j, 1.0);
    Matrix b = new Matrix(N, 1);
    b.set(0, 0, 1.0);
    x = B.solve(b);
    x.print(9, 6);
   } else {
    // If matrix is regular:
```

```
    // compute using 100 iterations of power method

    // initial guess for eigenvector
    x = new Matrix(N, 1, 1.0 / N);
    for (int i = 0; i < 100; i++) {
     x = A.times(x);
     // rescale
     x = x.times(1.0 / x.norm1());
    }
    x.print(9, 6);
   }
  }
}
```

Optimale probabilistische Strategie

Sprache: ampl (www.ampl.com)

Verwendeter Solver: minos

```
var a;var b;var c;var d;var e;var f;var p;
maximize prob: (3/37)*a+(28/37)*(b+c)+(18/37)*(d+e+f);
subject to one: a + b + c + d + e + f = 1;
subject to two:
        (9/37)*b*p+(28/37)*c*p+(19/37)*e*p+(18/37)*f*p = a;
subject to three:
        (3/37)*a*p+(9/37)*c*p+(18/37)*d*p+(19/37)*f*p = b;
subject to four:
        (34/37)*a*p+(28/37)*b*p+(19/37)*d*p+(18/37)*e*p = c;
subject to five:
        (9/37)*b*(1-p)+(28/37)*c*(1-p)+
        (19/37)*e*(1-p)+(18/37)*f*(1-p) = d;
subject to six:
        (3/37)*a*(1-p)+(9/37)*c*(1-p)+
        (18/37)*d*(1-p)+(19/37)*f*(1-p) = e;
subject to seven:
        (34/37)*a*(1-p)+(28/37)*b*(1-p)+
        (19/37)*d*(1-p)+(18/37)*e*(1-p) = f;
option solver minos;
solve;
display p;
```

```
MINOS 5.51: optimal solution found.
16 iterations, objective 0.506232105
Nonlin evals: constrs = 35, Jac = 34.
p = 0.643132
```

Imitation und Kooperation im Netzwerk

Sprache: Java

Verwendete Bibliotheken: java.util.*

```
public enum Strategy {
        COOP,
        DEF;
}
```

```
public class Agent {

                /*
                 * the agent's strategy can either be
                 * to cooperate or to defect
                 */
                private Strategy strategy;

                /*
                 * the strategy that the agent will
                 * adopt in the next round
                 */
                private Strategy nextStrategy;

                /*
                 * how successful was the agent in
                 * the last round?
                 */
                private double success;

                public Agent(double p) {
                        /*
                         * parameter p corresponds to the
                         * probability that the agent has
                         * a defective strategy when
                         * initialized
                         */
                        if(Math.random() < p)
```

```
                        strategy = Strategy.DEF;
                else
                        strategy = Strategy.COOP;
        }

        public void setStrategy(Strategy strategy) {
                this.strategy = strategy;
        }

        public Strategy getStrategy() {
                return this.strategy;
        }

        public double getSuccess() {
                return this.success;
        }

        public void adaptSuccess(double a) {
                this.success += a;
        }

        public void resetSuccess() {
                this.success = 0d;
        }

        public void setNextStrategy(Strategy s) {
                this.nextStrategy = s;
        }

        public void updateStrategy() {
                this.strategy = this.nextStrategy;
                this.nextStrategy = null;
        }

}
```

```java
public class AgentPair {
        private Agent a;
        private Agent b;

        public AgentPair(Agent a, Agent b) {
                this.a = a;
                this.b = b;
        }

        /* are the two agents of this pair identical to
         * agent x and agent y?
         * */
        public boolean equu(Agent x, Agent y) {
                if(a == null || b == null || x == null ||
                        y == null)
                        return false;
                return (a == x && b == y) || (b == x && a == y);
        }
}
```

```java
public class DilemmaGame {
        /*
         * sucker payoff
         */
        private double s;

        /*
         * temptation payoff
         */
        private double t;

        /*
         * a dilemma game is entirely determined
         * by its sucker and temptation payoff
         */
```

```
    public DilemmaGame(double s, double t) {
        assert -1d < s && s < 1d;
        assert 0 < t && t < 2d;
        assert s < t;
        this.s = s;     // sucker
        this.t = t; // temptation
    }

    /*
     * when two agents gamble, each of them
     * receives a payoff that depends on
     * both agents' strategies...
     */
    public DoublePair gamble(Agent x, Agent y) {
        if(x == null || y == null)
            return null;
        assert x != null && y != null;
        if(!x.getStrategy().toString().equals("COOP")
                    && !x.getStrategy().toString().
                            equals("DEF"))
            return null;
        assert x.getStrategy().toString().equals("COOP")
            || x.getStrategy().toString().
            equals("DEF");
        if(!y.getStrategy().toString().equals("COOP")
                    && !y.getStrategy().toString().
                            equals("DEF"))
            return null;
        assert y.getStrategy().toString().equals("COOP")
            || y.getStrategy().toString().
            equals("DEF");
        // if both cooperate, each receives payoff of 1
        if(x.getStrategy().toString().equals("COOP") &&
                    y.getStrategy().toString().
                            equals("COOP")) {
            return new DoublePair(1d,1d);
```

```
            }
            /*
             *  if one agent cooperates and the other
             *  defects, the agent who cooperates gets
             *  the sucker payoff and the agent who
             *  defects gets the temptation payoff
             */
            if(x.getStrategy().toString().equals("COOP") &&
                        y.getStrategy().toString().
                              equals("DEF")) {
                  return new DoublePair(s,t);
            }
            if(x.getStrategy().toString().equals("DEF") &&
                        y.getStrategy().toString().
                              equals("COOP")) {
                  return new DoublePair(t,s);
            }
            // if both agents defect, they both get 0
            if(x.getStrategy().toString().equals("DEF") &&
                        y.getStrategy().toString().
                              equals("DEF")) {
                  return new DoublePair(0d,0d);
            }
            return null;
      }

      public double getS() {
            return this.s;
      }

      public double getT() {
            return this.t;
      }

}
```

```
public interface NumbPair {

        Number getA();
        Number getB();

}
```

```
public class DoublePair implements NumbPair {

        private Double a;
        private Double b;

        public DoublePair(double a, double b) {
                this.a = a;
                this.b = b;
        }

        @Override
        public Number getA() {
                return this.a;
        }

        @Override
        public Number getB() {
                return this.b;
        }

}
```

```
public class IntPair implements NumbPair {

        private Integer a;
        private Integer b;

        public IntPair(int a, int b) {
                this.a = a;
```

```
			this.b = b;
		}

		@Override
		public Integer getA(){
			return a;
		}

		@Override
		public Integer getB() {
			return b;
		}

		public boolean equ(int x, int y) {
			return (a == x && b == y) ||
						(a == y && b == x);
		}

}
```

```
public abstract class Network {

	public abstract void setSize(int size);
	public abstract void setDensity(double density);
	public abstract void setRo(double ro);
	public abstract void setGame(double s, double t);
	public abstract void setP(double p);
	public abstract void build();
	public abstract void populate();
	public abstract void playRound();
	public abstract void updateStrategy();
	public abstract double getCoopFrac();
	public abstract void resetSuccess();

	/*
	 * returns modulo according to mathematical
```

```
	 * definition
	 */
	public int modulo(int x, int modulus) {
		if(modulus <= 0)
			return -1;
		assert modulus > 0;
		return ((x%modulus)+modulus)%modulus;
	}

	/*
	 * method gets array with probability
	 * density function over an interval of
	 * integers going from 0 to array length
	 * minus 1 and, according to
	 * this pdf, draws a number between
	 * 0 and array length minus 1.
	 */
	public int getRndm(double[] pdf) {
		double kSum = kahanSum(pdf);
		if(kSum < 0.99 || kSum > 1.01) {
			System.out.println("Sum is not 1 but "
				+kSum);
			return -1;
		}
		assert kahanSum(pdf) > 0.99 && kahanSum(pdf)
			< 1.01;
		double rndm = Math.random();
		int counter = 0;
		double sum = 0d;
		while(counter < pdf.length) {
			sum += pdf[counter];
			if(sum > 1.01 || sum < -0.01)
				break;
			if(rndm <= sum)
				return counter;
			counter++;
```

```
                }
                System.out.println("Something went wrong "
                                + "while drawing a random"
                                + " number. " + "Sum = " + sum);
                return -1;
        }

        /* returns the sum over a double array and is
         * more accurate than the trivial sum-over-
         * double-array approach
         */
        public double kahanSum(double[] x) {
                double sum = 0d;
                double c = 0d;
                for(int i = 0; i < x.length; i++) {
                        double y = x[i] - c;
                        double t = sum + y;
                        c = (t - sum) - y;
                        sum = t;
                }
                return sum;
        }

        /*
         * returns the difference of two double
         * numbers
         */
        public double getDiff(double a, double b) {
                return Math.max(a,b) - Math.min(a,b);
        }

}
```

```
import java.util.LinkedList;
import java.util.ListIterator;

public class LatticeNetwork extends Network {

        private int size;
        // number of vertices

        private int gridSize;
        // side length of square lattice

        private double ro;
        // prob. for unconditional imitation rule

        private double p;
        // prob. for agent to have defective strategy

        private Vertex[] vertices;

        private DilemmaGame game;
        // the game that the agents in the
        // network play

        private double pCopy;
        // prob. that an updating agent copies
        // their neighbor when playing replicator
        // rule

        /*
         * each agent plays a dilemma
         * game against all its neighbors
         */
        @Override
        public void playRound() {
                DoublePair payoffs = null;
```

```
        for(int i = 0; i < this.size; i++) {
            Agent currentAgent = this.vertices[i].
                getAgent();
            ListIterator<Vertex> iter =
            this.vertices[i].getneighbors()
                .listIterator();
            while(iter.hasNext()) {
                Vertex n = iter.next();
                if(!n.matched(this.vertices[i])
                && !this.vertices[i].matched(n))
                {
                    Agent newAgent =
                        n.getAgent();
                    payoffs = this.game.
                    gamble(currentAgent,
                    newAgent);
                    currentAgent.
                    adaptSuccess(
                    (double)payoffs.getA());
                    newAgent.adaptSuccess(
                    (double)payoffs.getB());
                    n.addMatched(this.
                        vertices[i]);
                    this.vertices[i].
                        addMatched(n);
                }
            }
        }
        for(int i = 0; i < this.size; i++) {
            vertices[i].resetMatched();
        }
    }

    /*
     * get the fraction of agents that
     * have a cooperative strategy at the moment
```

```
     */
    @Override
    public double getCoopFrac() {
        int cNumb = 0;
        for(int i = 0; i < this.size; i++)
            if(this.vertices[i].getAgent().
            getStrategy().toString().
            equals("COOP"))
                cNumb++;
        return (double)cNumb / this.size;
    }

    @Override
    public void populate() {
        for(int i = 0; i < this.size; i++)
            this.vertices[i].populate(new Agent(
                this.p));
    }

    /*
     * each agent updates their strategy
     * depending on some probability for the update rule
     * used and the success of his neighbors.
     */
    @Override
    public void updateStrategy() {
        for(int i = 0; i < this.size; i++) {
            if(Math.random() < this.ro) {
                // unconditional imitation -->
                // imitate the most successful
                // neighbor
                Agent msNeighbor =
                this.vertices[i].getAgent();
                ListIterator<Vertex> iter =
                this.vertices[i].getneighbors().
                listIterator();
```

```
                        while(iter.hasNext()) {
                                Agent nextAgent =
                                iter.next().getAgent();
                                if(nextAgent.
                                getSuccess() >
                                msNeighbor.
                                getSuccess())
                                        msNeighbor =
                                        nextAgent;
                        }
                        this.vertices[i].getAgent().
                        setNextStrategy(msNeighbor.
                        getStrategy());
                } else {
                        LinkedList<Vertex> neighbors =
                        this.vertices[i].getneighbors();
                        Vertex rndmNeighbor =
                        neighbors.get(
                        (int)(Math.random() *
                        neighbors.size()));
                        if(vertices[i].getAgent().
                        getSuccess() < rndmNeighbor.
                        getAgent().getSuccess()) {
                                double piN =
                                rndmNeighbor.getAgent().
                                getSuccess() /
                                rndmNeighbor.
                                getneighbors().size();
                                double piV =
                                vertices[i].getAgent().
                                getSuccess() /
                                vertices[i].
                                getneighbors().size();
                                double diff =
                                super.getDiff(piN, piV);
                                if(piN < piV)
```

```
                        diff = -diff;
                this.pCopy = 1d/2 +
                (1d/2)*(diff /
                super.getDiff(
                Math.max(this.game.
                getT(),1d),
                Math.min(this.game.
                getS(),0d)));
                if(this.pCopy < 0 ||
                this.pCopy > 1) {
                        System.out.
                        println(
                        "Error, pCopy "
                        + "out of "
                        + "range: "
                        + this.pCopy);
                }
                if(Math.random() <
                        this.pCopy)
                        this.vertices[i]
                        .getAgent().
                        setNextStrategy(
                        rndmNeighbor.
                        getAgent().
                        getStrategy());
                else
                        this.vertices[i]
                        .getAgent().
                        setNextStrategy(
                        this.vertices[i]
                        .getAgent().
                        getStrategy());
        } else {
                this.vertices[i].
                getAgent().
                setNextStrategy(
```

```
                                        this.vertices[i].
                                        getAgent().
                                        getStrategy());
                                }
                        }
                }
                for(int i = 0; i < this.size; i++) {
                        Agent uAgent = this.vertices[i].
                                getAgent();
                        uAgent.updateStrategy();
                        uAgent.resetSuccess();
                }
        }


        /*
         * build a lattice network
         */
        @Override
        public void build() {
                if(this.gridSize <= 0) {
                        System.out.println("Size must "
                        + "be positive.");
                        return;
                }
                assert gridSize > 0;
                if(this.p < 0 || this.p > 1) {
                        System.out.println("p must be between "
                        + "0 and 1.");
                        return;
                }
                assert 0 <= this.p && this.p <= 1;
                Vertex[][] lattice = new Vertex[this.gridSize]
                [this.gridSize];
                this.vertices = new Vertex[this.size];
                for(int i = 0; i < this.gridSize; i++) {
```

```
					for(int j = 0; j < this.gridSize; j++) {
						lattice[i][j] =
						this.vertices[
						i * this.gridSize + j] =
						new Vertex();
					}
				}
				for(int i = 0; i < this.gridSize; i++) {
					for(int j = 0; j < this.gridSize; j++) {
						for(int x = -1; x < 2; x++) {
							loop: for(int y = -1;
								y < 2; y++) {
								if(x == 0 &&
									y == 0)
									continue
									loop;
								this.vertices[
								i*this.gridSize
								+ j].
								addNeighbour(
								lattice[
								modulo(
								i+x,this.
								gridSize)]
								[modulo(
								j+y,this.
								gridSize)]);
							}
						}
					}
				}
				this.populate();
			}

			@Override
			public void setGame(double s, double t) {
```

```
            game = new DilemmaGame(s,t);
    }

    @Override
    public void setP(double p) {
            if(p < 0 || p > 1)
                    return;
            assert p >= 0 && p <= 1;
            this.p = p;
    }

    @Override
    public void setSize(int size) {
            if(Math.pow((int)Math.sqrt(size), 2) != size) {
                    // Size of lattice network must be a
                    // square number. Size is automatically
                    // rounded off to next
                    // square integer.
                    size = (int)Math.pow(
                    (int)Math.sqrt(size), 2);
            }
            // size is a square integer
            this.size = size;
            this.gridSize = (int)(Math.sqrt(size));
    }

    public int getSize() {
            return this.size;
    }

    @Override
    public void setDensity(double dentsity) {
            // do nothing;
            // each vertex in the network has
            // 8 neighbors
    }
```

```
	@Override
	public void setRo(double ro) {
		if(ro < 0 || ro > 1) {
			System.out.println("ro must be between "
					+ "0 and 1.");
			return;
		}
		assert 0 <= ro && ro <= 1;
		this.ro = ro;
	}


	@Override
	public String toString() {
		String output = new String("");
		for(int i = 0; i < this.size; i++) {
			if(i % this.gridSize == 0)
				output += "\r\n";
			output += (vertices[i].getAgent().
			getStrategy().toString().
			equals("COOP") ?
			"+" : "o") + " ";
		}
		return output + "\r\n";
	}

	public String toVal() {
		String output = new String("");
		for(int i = 0; i < this.size; i++) {
			if(i % this.gridSize == 0)
				output += "\r\n";
			output += Math.round(vertices[i].
			getAgent().
			getSuccess()*10)/10d + " ";
		}
```

```
                return output + "\r\n";
        }

        @Override
        public void resetSuccess() {
                for(int i = 0; i < this.size; i++)
                        vertices[i].getAgent().resetSuccess();
        }
}
```

```
import java.util.Comparator;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.ListIterator;

// Scale free network
public class SFNetwork extends Network {

        private int size;
        // number of vertices

        private double density;
        // avg. number of edges between vertices

        private double ro;
        // prob. for unconditional imitation rule

        private double p;
        // prob. for agent to have defective strategy

        private DilemmaGame game;
        // the game that the agents in the network play

        private Vertex[] vertices;
        // vertices of network

```

```
	private int[] noN;
	// number of neighbors of each vertex

	private double[] pdf;
	// fraction of connections per vertex to
	// total number of connections


	@Override
	public void setSize(int size) {
		this.size = size;
	}

	@Override
	public void setDensity(double density) {
		this.density = density;

	}

	@Override
	public void setRo(double ro) {
		this.ro = ro;

	}

	@Override
	public void setGame(double s, double t) {
		this.game = new DilemmaGame(s,t);

	}

	@Override
	public void setP(double p) {
		this.p = p;

	}
```

```java

	/*
	 *
	 * build a scale free network according to
	 * the  Barabasi-Albert model
	 */
	@Override
	public void build() {
		this.vertices = new Vertex[this.size];
		this.noN = new int[this.size];
		this.pdf = new double[this.size];
		for(int i = 0; i < this.vertices.length; i++) {
			this.vertices[i] = new Vertex();
		}
		int counter = 1;
		// start with 'density' initially
		// connected vertices
		while(counter < this.density) {
			int rndm = (int)(Math.random() *
				counter);
			assert rndm < counter;
			this.vertices[counter].
			addNeighbourMutually(this.
				vertices[rndm]);
			this.noN[counter++]++;
			this.noN[rndm]++;
		}
		// add vertices to the network and connect
		// each one to 'density' existing nodes
		// according to the power law
		while(counter < this.size) {
			for(int i = 0; i < this.density; i++) {
				updatePdf(counter);
				int rndm = 0;
				do {
					rndm = this.
```

```
                                        getRndm(pdf);
                        } while(this.vertices[counter].
                                getneighbors().contains(
                                this.vertices[rndm]) ||
                                this.vertices[rndm].
                                getneighbors().
                                contains(this.
                                vertices[counter]));
                        // draw new rndm while
                        // connection already exists
                        this.vertices[counter].
                        addNeighbourMutually(this.
                                vertices[rndm]);
                        this.noN[counter]++;
                        this.noN[rndm]++;
                    }
                    counter++;
            }
            this.populate();

            assert !allNDist();

    }

    private void updatePdf(int k) {
            int noC = 0;    // total number of connections
            for(int i = 0; i < noN.length; i++) {
                    if(!this.vertices[i].getneighbors().
                    contains(this.vertices[k]) && i != k)
                            noC += this.noN[i];
            }
            for(int i = 0; i < this.pdf.length; i++) {
                    if(!this.vertices[i].getneighbors().
                    contains(vertices[k]) && i != k)
                            this.pdf[i] =
                            (double)this.noN[i] / noC;
```

```
                        else
                                this.pdf[i] = 0d;
                }
        }


        /*
         * each agent plays a dilemma
         * game against all its neighbors
         */
        @Override
        public void playRound() {
                DoublePair payoffs = null;
                for(int i = 0; i < this.size; i++) {
                        Agent currentAgent = this.vertices[i].
                                getAgent();
                        ListIterator<Vertex> iter =
                        this.vertices[i].getneighbors().
                                listIterator();
                        while(iter.hasNext()) {
                                Vertex n = iter.next();
                                if(!n.matched(this.vertices[i])
                                && !this.vertices[i].matched(n))
                                {
                                        Agent newAgent =
                                                n.getAgent();
                                        payoffs = this.game.
                                        gamble(
                                        currentAgent, newAgent);
                                        currentAgent.
                                        adaptSuccess((double)
                                                payoffs.getA());
                                        newAgent.
                                        adaptSuccess((double)
                                                payoffs.getB());
                                        n.addMatched(this.
```

```
                                                vertices[i]);
                                        this.vertices[i].
                                                addMatched(n);
                                }
                        }
                }
                for(int i = 0; i < this.size; i++) {
                        vertices[i].resetMatched();
                }
        }

        /*
         * each agent updates its strategy
         * depending on some probability for the update rule
         * used and the success of his neighbors.
         */
        @Override
        public void updateStrategy() {
                for(int i = 0; i < this.size; i++) {
                        if(Math.random() < this.ro) {
                                // unconditional imitation -->
                                // imitate the most successful
                                // neighbor
                                Agent msNeighbor = this.
                                vertices[i].getAgent();
                                ListIterator<Vertex> iter =
                                this.vertices[i].
                                getneighbors().listIterator();
                                while(iter.hasNext()) {
                                        Agent nextAgent = iter.
                                        next().getAgent();
                                        if(nextAgent.
                                        getSuccess() >
                                        msNeighbor.getSuccess())
                                                msNeighbor =
                                                nextAgent;
```

```
                }
                this.vertices[i].getAgent().
                setNextStrategy(
                msNeighbor.getStrategy());
            } else {
                LinkedList<Vertex> neighbors =
                this.vertices[i].getneighbors();
                Vertex rndmNeighbor = neighbors.
                get((int)(Math.random() *
                    neighbors.size()));
                if(vertices[i].getAgent().
                getSuccess() < rndmNeighbor.
                getAgent().getSuccess()) {
                    double piN =
                    rndmNeighbor.
                    getAgent().
                    getSuccess() /
                    rndmNeighbor.
                    getneighbors().
                    size();
                    double piV =
                    vertices[i].
                    getAgent().
                    getSuccess() /
                    vertices[i].
                    getneighbors().
                    size();
                    double diff =
                    super.getDiff(piN, piV);
                    if(piN < piV)
                        diff = -diff;
                    double pCopy = 1d/2 +
                    (1d/2)*(diff /
                    super.getDiff(
                    Math.max(this.game.
                    getT(),1d),
```

```
                                        Math.min(this.game.
                                        getS(),0d)));
                                        if(pCopy < 0 ||
                                                pCopy > 1)
                                                System.out.
                                                println(
                                                "Error, pCopy "
                                                + "out of "
                                                + "range: "
                                                + pCopy);
                                        if(Math.random() <
                                                        pCopy)
                                                this.vertices[i]
                                                .getAgent().
                                                setNextStrategy(
                                                rndmNeighbor.
                                                getAgent().
                                                getStrategy());
                                        else
                                                this.vertices[i]
                                                .getAgent().
                                                setNextStrategy(
                                                this.vertices[i]
                                                .getAgent().
                                                getStrategy());
                                } else {
                                        this.vertices[i].
                                        getAgent().
                                        setNextStrategy(
                                        this.vertices[i].
                                        getAgent().
                                        getStrategy());
                                }
                        }
                }
                for(int i = 0; i < this.size; i++) {
```

```
                        Agent uAgent = this.vertices[i].
                        getAgent();
                        uAgent.updateStrategy();
                        uAgent.resetSuccess();
                }
        }

        /*
         * get the fraction of agents that
         * have a cooperative strategy at the moment
         */
        @Override
        public double getCoopFrac() {
                int cNumb = 0;
                for(int i = 0; i < this.size; i++)
                        if(this.vertices[i].getAgent().
                        getStrategy().toString().
                        equals("COOP"))
                                cNumb++;
                return (double)cNumb / this.size;
        }

        private boolean allNDist() {
                for(int i = 0; i < this.size; i++) {
                        this.vertices[i].getneighbors().sort(
                        new Comparator<Vertex>() {

                                @Override
                                public int compare(Vertex v1,
                                Vertex v2) {
                                        return v1.getIndex() <
                                        v2.getIndex() ? 1 :
                                        v1.getIndex() >
                                        v2.getIndex() ? -1 : 0;
                                }

```

```java
                        });
                        Iterator<Vertex> iter = this.
                        vertices[i].getneighbors().iterator();
                        Vertex actVert = iter.next();
                        while(iter.hasNext()) {
                                Vertex nextVert = iter.next();
                                if(actVert == nextVert) {
                                        return false;
                                }
                                actVert = nextVert;
                        }
                }
                return true;
        }

        @Override
        public void populate() {
                for(int i = 0; i < this.size; i++)
                        this.vertices[i].populate(new Agent(
                                this.p));
        }

        @Override
        public void resetSuccess() {
                for(int i = 0; i < this.size; i++)
                        vertices[i].getAgent().resetSuccess();
        }

}
```

```java
import java.util.LinkedList;
import java.util.ListIterator;

public class Vertex {

        private static int instanceCounter;
```

```
	// counts the instances of 'Vertex' created so far

	private Agent agent;
	// a vertex has a unique agent

	private LinkedList<Vertex> neighbors;
	// a vertex has 1-* neighbors

	private int index;
	// a vertex has a unique index

	private LinkedList<Vertex> matched;
	// list of neighbors that this vertex'
	// agent played with in the current round

	public Vertex() {
		this.index = instanceCounter++;
		this.neighbors = new LinkedList<Vertex>();
	}

	public boolean addNeighbourMutually(Vertex v) {
		return this.neighbors.add(v) &&
				v.addNeighbour(this);
	}

	public boolean addNeighbour(Vertex v) {
		return this.neighbors.add(v);
	}

	public void populate(Agent agent) {
		this.agent = agent;
	}

	public LinkedList<Vertex> getneighbors() {
		return this.neighbors;
	}
```

```java

	public int getIndex() {
		return this.index;
	}

	public Agent getAgent() {
		return this.agent;
	}

	public void addMatched(Vertex v) {
		if(this.matched == null)
			this.matched = new LinkedList<Vertex>();
		this.matched.add(v);
	}

	public boolean matched(Vertex v) {
		if(this.matched == null)
			return false;
		return this.matched.contains(v);
	}

	public void resetMatched() {
		this.matched = new LinkedList<Vertex>();
	}



	@Override
	public boolean equals(Object o) {
		if(o == null)
			return false;
		if(!(o instanceof Vertex))
			return false;
		Vertex v = (Vertex)o;
		// index is a unique identifier
		return this.index == v.getIndex();
```

```
	}

	public int getEdgeCount() {
		return neighbors.size();
	}



	public void play(DilemmaGame game) {
		ListIterator<Vertex> iter =
				neighbors.listIterator();
		while(iter.hasNext()) {
			Vertex nextVertex = iter.next();
			if(nextVertex.getIndex() > this.index) {
				DoublePair db = game.gamble(
				this.agent, nextVertex.
				getAgent());
				this.agent.adaptSuccess(
				(double)db.getA());
				nextVertex.getAgent().
				adaptSuccess(
				(double)db.getB());
				nextVertex.play(game);
			}
		}
	}


	/*
	 * return the most successful neighbour from
	 * the neighbourhood
	 */
	public Agent getMSNeighbour() {
		Agent ms = this.agent;
		ListIterator<Vertex> iter = neighbors.
			listIterator();
```

```
            while(iter.hasNext()) {
                    Agent nextAgent = iter.next().
                            getAgent();
                    if(nextAgent.getSuccess() > this.agent.
                                    getSuccess())
                            ms = nextAgent;
            }
            return ms;
    }

}
```

Clusterbildung im Netzwerk (Ausschnitt)

Netzwerkart: quadratisches Gitternetzwerk
Netzwerkgröße: 10 000
Randbehandlung: periodisch
Spiel: Gefangenendilemma (S = -0.7, T = 1.1)
rho: 0.5
Anteil kooperativer Agenten zu Beginn: 0.5
Netzwerkzustand nach Iterationen: n = 0, 1, 2, 3, 4, 5, 6, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200
o . . . betrügerischer Agent
+ . . . kooperierender Agent
Ausschnitt des Gesamtnetzwerks